МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритм Ахо-Корасик»

Тема: Поиск с возвратом

Студент гр. 8304 _____ Птухов Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Построение и анализ алгоритма Ахо-Корасик на основе реализации точного множественного поиска.

Вариант 1.

Основные теоретические положения.

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 тексте T.

Например, образец ab??c?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 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, 100000T,$1 \le |T| \le 100000$)

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.

Описание алгоритма.

Для решения поставленной задачи были реализованы функции по созданию бора и создания конечного детерминированного автомата по нему. Исходный шаблон был разделен на k подстрок разделенных символами джокера. Каждая из данных подстрок была добавлена в бор. Был создан массив C длина которого равна длине ранее считанного текста. Данный массив хранит кол-во подстрок шаблона, который могут находить с i-ой позиции. Если значение c[i] равно k, то с i-ой позиции начинается шаблон в исходном тексте. Заполнение данного массива осуществляется при помощи посимвольного чтения текста и соответствующего перехода по ранее созданному автомату.

Описание основных структур данных и функций.

1) Структура Vertex, хранящая в себе: массив next – доступные переходы из данной вершины (переходы по автомату), is_leaf – является ли данная вершина листом, str_nums – индексы подстрок завершающихся в данной вершине, link – суффиксная ссылка, from – номер вершины предка, how – символ перехода из предка в текущую вершину.
2) Make_bor_vertex – функция создающая вершину бора.
3) addToBor – функция осуществляющая добавления строки в бор. Данная подзадача реализована при помощи посимвольного прохождения по строке и бору.
4) Взаимно рекурсивные функции go и get_link – функции осуществляющие перехода по ребру или по суффиксной ссылке для ранее принятой вершины u.

Тестирование.

Таблица 1 – Результаты тестирование

Ввод	**Вывод**
ACT A$ $ T	1
ACT AC$ $ T	Empty result
NACGNTTACGGTCACNN AC$$T$AC$$ $ H	2
AACGTGTNN ACGT $ C	2
ACCATTACG A$$ C	4

Вывод.

В ходе работы был построен и анализирован алгоритм Ахо-Корасик на основе решения задачи о реализации множественного поиска. Исходный код программы представлен в приложении А.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

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

struct Vertex
{
    std::vector<int> next;
    bool is_leaf = false;
    std::vector<size_t> str_nums;

    int link = -1;
    int from = -1;
    char how = 0;
    std::vector<int> go;
};

Vertex make_bor_vertex(int from, char how)
{
    Vertex vert;
    vert.next = { -1, -1, -1, -1, -1 };
    vert.go = { -1, -1, -1, -1, -1 };
    vert.from = from;
    vert.how = how;

    return vert;
}

void addToBor(std::string& str, std::vector<Vertex>& bor, std::map<char, int>&
alphabet, int str_num)
{
    int borInd = 0;
    for (auto c : str)
    {
        char cInd = alphabet[c];

        if (bor[borInd].next[cInd] == -1)
        {
            bor.push_back(make_bor_vertex(borInd, cInd));
            bor[borInd].next[cInd] = bor.size() - 1;
        }

        borInd = bor[borInd].next[cInd];
    }

    bor[borInd].is_leaf = true;
    bor[borInd].str_nums.push_back(str_num);
}

int go(int v, char c, std::vector<Vertex>& bor);

int get_link(int v, std::vector<Vertex>& bor)
{
    if (bor[v].link == -1)
    {
        if (v == 0 || bor[v].from == 0)
            bor[v].link = 0;
```

```
        else
            bor[v].link = go(get_link(bor[v].from, bor), bor[v].how, bor);
    }

    return bor[v].link;
}

int go(int v, char c, std::vector<Vertex>& bor)
{
    if (bor[v].go[c] == -1)
    {
        if (bor[v].next[c] != -1)
            bor[v].go[c] = bor[v].next[c];
        else
        {
            if (v == 0)
                bor[v].go[c] = 0;
            else
                bor[v].go[c] = go(get_link(v, bor), c, bor);
        }
    }

    return bor[v].go[c];
}

void AXO_CORASIK(std::istream& in, std::ostream& out)
{
    std::map<char, int> alphabet;
    alphabet['A'] = 0;
    alphabet['C'] = 1;
    alphabet['G'] = 2;
    alphabet['T'] = 3;
    alphabet['N'] = 4;

    std::string text;
    std::string pattern;
    char J;

    in >> text;
    in >> pattern;
    in >> J;

    char no;
    in >> no;

    pattern += J;

    std::vector<std::string> q;
    std::vector<size_t> l;

    std::string cur;
    for (size_t i = 0; i < pattern.length(); ++i)
    {
        if (pattern[i] == J)
        {
            if (!cur.empty())
            {
                q.push_back(cur);
                l.push_back(i - cur.size() + 1);
            }
            cur.clear();
        }
        else
```

```
        cur += pattern[i];
}

std::vector<Vertex> bor;
bor.push_back(make_bor_vertex(0, 0));

out << "Pattern substr - ";
for (size_t i = 0; i < q.size(); ++i)
{
    addToBor(q[i], bor, alphabet, i);
    if (i == q.size() - 1)
        out << q[i];
    else
        out << q[i] << ", ";
}
out << "\n\nBor created\n";

std::vector<size_t> c(text.size());

bool f;
int vert_num;
int u = 0;
for (size_t i = 0; i < text.length(); ++i)
{
    out << "Go from " << u << " vertex";
    u = go(u, alphabet[text[i]], bor);
    out << " to " << u << " vertex\nLink way for this vertex to start - ";

    f = false;
    for (int v = u; v != 0; v = get_link(v, bor))
    {
        out << v << " -> ";

        if (bor[v].is_leaf)
        {
            f = true;
            vert_num = v;
            for (auto& str_num : bor[v].str_nums)
            {
                int j = i - q[str_num].length() + 1;
                if (j >= l[str_num] - 1)
                    ++c[j - l[str_num] + 1];
            }
        }
    }
    out << "0";

    if (f == true)
        out << "\nWas finded leaf - " << vert_num << " vertex";
    out << "\n\n";
}


for (size_t i = 0; i < text.size(); ++i)
    if (c[i] == q.size())
    {
        bool is_correct = true;
        for (size_t k = i; k < i + pattern.size() - 1; ++k)
        {
            if (pattern[k - i] == J && text[k] == no)
            {
                is_correct = false;
                break;
```

```
                }

            }
            if (is_correct)
                out << i + 1 << "\n";
        }
}

int main()
{
    int n1, n2;
    std::cout << "Read from: 1 - file, 2 - term\n";
    std::cin >> n1;
    std::cout << "\nWrite to: 1 - file, 2 - term\n";
    std::cin >> n2;

    std::ifstream in("input.txt");
    std::ofstream out("output.txt");

    if (n1 == 1 && n2 == 1)
        AXO_CORASIK(in, out);
    if (n1 == 1 && n2 == 2)
        AXO_CORASIK(in, std::cout);
    if (n1 == 2 && n2 == 1)
        AXO_CORASIK(std::cin, out);
    if (n1 == 2 && n2 == 2)
        AXO_CORASIK(std::cin, std::cout);

    return 0;

}
```