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

ОТЧЕТ

по практической работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студент гр. 8304 Воропаев А.О.

Преподаватель Размочаева Н.В.

Санкт-Петербург
2020

Вариант 2.

Цель работы.

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с *джокером*.

Расширение.

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

Основные теоретические положения.

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 неопределённой длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы. Все строки содержат символы из алфавита $\{A,C,G,T,N\}$

Описание алгоритма.

Сначала происходит построение бора, по безмасочным подстрокам входного шаблона. Бор в данной программе играет роль автомата, который будет использоваться при обработке текста. Данные о вершинах бора содержатся в структурах vertex. Затем текст, в котором необходимо найти заданный шаблон, обрабатывается по одному символу. Изначально мы находимся в корне бора, затем для очередного символа мы пытаемся перейти в состояние автомата соответствующее этому символу с помощью функции go, если оно рассчитано, иначе мы переходим по ссылке назад по бору. После получения вершины, отвечающей за следующее состояние, мы проходим по суффиксным ссылкам, пока не дойдем до корня бора. Если при проходе по ссылкам мы нашли вершину-лист, то увеличиваем соответствующее значение в векторе шаблонов templates_count. Если после очередного увеличения значения, она стало равно

кол-ву подстрок в изначальном шаблоне, то в этой позиции начинается искомый шаблон.

Сложность алгоритма.

Поиск подстрок заданного шаблона с помощью алгоритма Ахо-Корасик выполняется за время O(m+n+a), где n — суммарная длина подстрок, то есть длина шаблона, m — длина текста, a — количество появлений подстрок шаблона. Далее просто надо пробежаться по массиву templates_count и просмотреть значения ячеек за время O(m).

Описание основных структур данных и функций.

1. Структура, хранящая в себе информацию о вершинах бора.

```
struct vertex {
    int next[K] = {-1, -1, -1, -1, -1};
    bool leaf = false;
    int p = -1;
    char pch = 0;
    int link = -1;
    int go[K] = {-1, -1, -1, -1, -1};
    std::vector<int> str_nums;
};
```

next – массив, хранящий пути в боре для каждого символа

leaf – флаг, указывающий, является ли вершина листом

p – номер предыдущей вершины

pch – символ предыдущей вершины

link – суффиксная ссылка.

go – массив переходов по атомату.

str_nums – ветор, содержащий номера строк, заканчивающихся в вершине.

2. void init(std::vector<vertex>& bor, std::vector<std::pair<std::string, int>>& substrings, std::string& str, char joker) – функция для инициализации бора, вектора подстрок.
   bor – бор
   substrings – вектор подстрок
   str – шаблон
   joker – символ маски
3. void add_string (const std::string& s, int num, std::vector<vertex>& bor) – функция добавления строки в бор
   s – добавляемая строка
   num – номер строки
   bor – бор

4. int go (int v, char c, std::vector<vertex>& bor) – функция для перехода к следующему состоянию автомата по символу c.
   v – номер вершины, из которой осуществляется переход
   c – символ, по которому осуществляется переход
   bor – бор

5. int get_link (int v, std::vector<vertex>& bor) – функция для расчета ссылки от вершины под номером v
   v – номер вершины в боре
   bor – бор

6. void processing(std::vector<std::pair<std::string, int>>& substrings, std::vector<vertex>& bor, std::string& text, std::string& str) – функция, в которой осуществляется добавление подстрок шаблона в бор и посимвольная обработка текста, с последующим выводом результатов.
   substrings – вектор подстрок шаблона
   bor – бор

text – текст

str – шаблон

Тестирование.

```
AAAACAAG
AA$
$
Quantity of vertexes: 3
Substring has been found at the position #1
It has the following intersections:
_____________________________________________
Substring has been found at the position #2
It has the following intersections:
With the string on the pos #1(AAA->AAA)
_____________________________________________
Substring has been found at the position #3
It has the following intersections:
With the string on the pos #1(AAC->AAA)
With the string on the pos #2(AAC->AAA)
_____________________________________________
Substring has been found at the position #6
It has the following intersections:
_____________________________________________
```

Рисунок 1 – Результаты 1-ого теста

```
CGAATNCGCGTN
CG$
$
Quantity of vertexes: 3
Substring has been found at the position #1
It has the following intersections:
_____________________________________________
Substring has been found at the position #7
It has the following intersections:
_____________________________________________
Substring has been found at the position #9
It has the following intersections:
With the string on the pos #7(CGT->CGC)
_____________________________________________
```

Рисунок 2 – Результаты 2-го теста

Вывод.

Был получен опыт в реализации алгоритма Ахо-Корасик. Также в ходе работы была проанализирована сложность работы алгоритма.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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


const int K = 5; //Размер алфавита
std::map<char, int> alphabet;
char in, out;

//Структура, описывающая вершины бора
struct vertex {
    int next[K] = {-1, -1, -1, -1, -1};
    bool leaf = false;
    int p = -1;
    char pch = 0;
    int link = -1;
    int go[K] = {-1, -1, -1, -1, -1};
    std::vector<int> str_nums;
};

//Инициализация бора и вектора подстрок
void init(std::vector<vertex>& bor, std::vector<std::pair<std::string, int>>& substrings, std::string&
str, char joker) {
    vertex root = vertex();
    bor.push_back(root);


    std::pair<std::string, int> tmp;

    //Разрбиение строки с масками на подстроки без масок
    for(int i = 0; i < str.size(); ++i) {
        if(str[i] == joker) {
            if(!tmp.first.empty()) {
                tmp.second = i + 1 - tmp.first.length();
                substrings.push_back(tmp);
                tmp.first.clear();
            }
            continue;
        }

        tmp.first += str[i];
    }
    if(!tmp.first.empty()) {
        tmp.second = str.size() - tmp.first.length();
        substrings.push_back(tmp);
    }
}
```

```
//Функция добавления строки в бор
void add_string (const std::string& s, int num, std::vector<vertex>& bor) {
    int v = 0;
    vertex tmp = vertex();

    //Проход по каждому символу
    for (char i : s) {
        char c = alphabet[i];
        //Если очередной символ не найден в боре, то добавляем его
        if (bor[v].next[c] == -1) {
            tmp.p = v;
            tmp.pch = c;
            bor[v].next[c] = bor.size();
            bor.push_back(tmp);
        }
        v = bor[v].next[c];
    }
    bor[v].leaf = true;
    bor[v].str_nums.push_back(num);
}


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

//Функция для вычисления суффиксной ссылки
int get_link (int v, std::vector<vertex>& bor) {

    if (bor[v].link == -1) {
        if (v == 0 || bor[v].p == 0)
            bor[v].link = 0;
        else{
            bor[v].link = go(get_link(bor[v].p, bor), bor[v].pch, bor);
        }
    }
    return bor[v].link;
}

//Функция для перехода по состояниям автомата
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 processing(std::vector<std::pair<std::string, int>>& substrings, std::vector<vertex>& bor,
std::string& text, std::string& str) {

    //Добавление подстрок в бор
    for(int i = 0; i < substrings.size(); ++i) {
        add_string(substrings[i].first, i + 1 , bor);
    }

    //Создание вектора шаблонов
    std::vector<int> templates_count(text.size());
    templates_count.insert(templates_count.begin(), 0);


    int count = 0;
    vertex current = bor[0];

    std::ofstream output;
    if(out == 'c')
        std::cout << "Quantity of vertexes: " << bor.size() << std::endl;
    else if(out == 'f'){
        output.open("C:/Users/Anton/CLionProjects/PAA_LAB5/output");
        output << "Quantity of vertexes: " << bor.size() << std::endl;
    }

    for(int i = 0; i < text.size(); ++i) {
        char c = alphabet[text[i]];
        count = go(count, c, bor);//Получить номер вершины в боре

        //Проход по суффиксным ссылкам
        for (int v = count; v != 0; v = get_link(v, bor)) {
            if(bor[v].leaf)
                //Проход по подстрокам, которые оканчиваются в листе
                for(int j = 0; j < bor[v].str_nums.size(); ++j) {

                    int pos = i - substrings[bor[v].str_nums[j] - 1].first.length() + 1;
                    int t_index = pos - substrings[bor[v].str_nums[j] - 1].second + 1;

                    //Если позиция подстроки минус её позиция в строке с маской не отрицательное
число
                    if(t_index >= 0) {
                        templates_count[t_index] += 1;

                        //Если в определённой позиции в векторе шаблонов кол-во повторений = кол-ву
подстрок без масок, то шаблон найден
                        if (templates_count[t_index] == substrings.size()) {

                            if(out == 'c') {
                                std::cout << "Substring has been found at the position #" << t_index + 1 <<
```

```
std::endl <<
                        "It has the following intersections:\n";
                }
                else if(out == 'f'){
                    output << "Substring has been found at the position #" << t_index + 1 <<
std::endl <<
                        "It has the following intersections:\n";
                }


                //Поиск пересечений
                for(int k = t_index + 1; k < t_index + str.size() - 1; ++k) {
                    if(templates_count[k - str.size() + 1] == substrings.size()) {

                        if(out == 'c') {
                            std::cout << "With the string on the pos #" << k - str.size() + 2 << "("
                                << std::string(text.begin() + t_index,text.begin() + t_index +
str.size() - 1)
                                << "->"
                                << std::string(text.begin() + k - str.size() + 1, text.begin() + k)
                                << ")\n";
                        }
                        else if(out == 'f') {
                            output << "With the string on the pos #" << k - str.size() + 2 << "("
                                << std::string(text.begin() + t_index,text.begin() + t_index + str.size()
- 1)
                                << "->"
                                << std::string(text.begin() + k - str.size() + 1, text.begin() + k)
                                << ")\n";
                        }
                    }
                }
                if(out == 'c') {
                    std::cout << "_____\n";
                }
                else if(out == 'f'){
                    output << "_____\n";
                }
            }
        }
    }
}


int main() {

    //Инициалтзация алфавита
    alphabet['A'] = 0;
    alphabet['C'] = 1;
```

```
    alphabet['G'] = 2;
    alphabet['T'] = 3;
    alphabet['N'] = 4;



    std::cout << "Enter how you want to read and write data(c - console, f - file)\n";
    std::cin >> in >> out;
    if((out != 'c' && out != 'f') || (in != 'c' && in != 'f')) {
        std::cout << "Incorrect data\n";
        return 1;
    }

    std::string text;
    std::string str;
    char joker = 0;


    if(in == 'c') {
        std::cin >> text >> str >> joker;
    }
    else if(in == 'f') {
        std::ifstream input("C:/Users/Anton/CLionProjects/PAA_LAB5/input");
        input >> text >> str >> joker;
    }


    str += joker;
    std::vector<std::pair<std::string, int>> substrings;
    std::vector<vertex> bor;

    //Инициализация вектора подстрок и бора
    init(bor, substrings, str, joker);

    //Запуск алгоритма
    processing(substrings, bor, text, str);

    return 0;
}
```