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

Студентка гр. 8304 _____ Мельникова О.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритм Ахо-Корасик для оптимального поиска всех вхождений данных подстрок в строку.

Задание.

1. Разработайте программу, решающую задачу точного поиска набора образцов.

   Вход:

   Первая строка содержит текст $(T, 1 \le |T| \le 100000)$.

   Вторая — число n $(1 \le |n| \le 3000)$, каждая следующая из n строк содержит шаблон из набора $P = \{p_1, \ldots, p_n\} 1 \le |p_i| \le 75$.

   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, решите 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 T.

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

Вход:

Текст $(T, 1 \le |T| \le 100000)$.

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.

Дополнительное задание (вариант 2): Подсчитать количество вершин в автомате; вывести список найденных образцов, имеющих пересечение с другими найденными образцами в строке поиска.

Описание алгоритма.

Бор — это дерево, в котором каждая вершина обозначает какую-то строку (корень обозначает нулевую строку — ε). На ребрах между вершинами написана 1 буква (в этом его принципиальное различие с суффиксными деревьями и др.), таким образом, добираясь по ребрам из корня в какую-нибудь вершину и контангенируя буквы из ребер в порядке обхода, мы получим строку,

соответствующую этой вершине. Строить бор будем последовательным добавление исходных строк. Изначально у нас есть 1 вершина, корень (root) — пустая строка. Добавление строки происходит так: начиная в корне, двигаемся по нашему дереве, выбирая каждый раз ребро, соответствующее очередной букве строки. Если такого ребра нет, то мы создаем его вместе с вершиной. Бор играет роль автомата, который будет использоваться при обработке текста. Обрабатываться он будет посимвольно, из корня мы стараемся перейти в состояние автомата, соответствующее этому символу, если оно не рассчитано возвращаемся по суффиксной ссылке. После получения вершины, отвечающей за следующее состояние, проходим до корня бора по ссылкам. Если на пути мы нашли вершину-лист, то значение в векторе шаблонов увеличивается. При условии, что оно равно количеству подстрок в инзначальном шаблоне, в этой позиции и начинается искомый шаблон.

Сложность алгоритма O(m+n+a), где m — длина текста, n — общая длина подстрок, a — количество вхождений.

Тестирование.

Входные данные: CCCA 1 CC
Выходные данные: Элемент текста: C Состояние: 0 Новое состояние: 1 Возврат к корню. Текущее состояние: 1 Элемент текста: C Состояние: 1 Новое состояние: 2 Возврат к корню. Текущее состояние: 2 Лист!!! Позиция в тексте: 1 Номер образца: 1 Возврат к корню. Текущее состояние: 1 Элемент текста: C Состояние: 2

Новое состояние: 2
    Возврат к корню. Текущее состояние: 2
        Лист!!! Позиция в тексте: 2 Номер образца: 1
    Возврат к корню. Текущее состояние: 1
Элемент текста: A Состояние: 2
Новое состояние: 0

Входные данные:
ACTANCA
A$$A$
$

Выходные данные:
Количество вершин: 2
Номер вершины в боре: 1
    Проход по суффиксным ссылкам:
    Позиция подстроки(A): 0; Позиция в строке с маской(B): 2; A-B = 0
    Позиция подстроки(A): 0; Позиция в строке с маской(B): 5; A-B = -3
Номер вершины в боре: 0
    Проход по суффиксным ссылкам:
Номер вершины в боре: 0
    Проход по суффиксным ссылкам:
Номер вершины в боре: 1
    Проход по суффиксным ссылкам:
    Позиция подстроки(A): 3; Позиция в строке с маской(B): 2; A-B = 3
    Позиция подстроки(A): 3; Позиция в строке с маской(B): 5; A-B = 0
    Количество повторений в векторе шаблонов равно количеству подстрок без масок. Шаблон найден.
    Индекс вхождения: 1
Пересечения с другими найденными образцами:
Номер вершины в боре: 0
    Проход по суффиксным ссылкам:
Номер вершины в боре: 0
    Проход по суффиксным ссылкам:
Номер вершины в боре: 1
    Проход по суффиксным ссылкам:
    Позиция подстроки(A): 6; Позиция в строке с маской(B): 2; A-B = 6
    Позиция подстроки(A): 6; Позиция в строке с маской(B): 5; A-B = 3
    Количество повторений в векторе шаблонов равно количеству подстрок без масок. Шаблон найден.
    Индекс вхождения: 4

Пересечения с другими найденными образцами:
ANCA->ACTAN

Вывод.

В результате лабораторной работы были изучены алгоритм Ахо-Корасик для поиска вхождений нескольких подстрок в строку.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

```
#INCLUDE <IOSTREAM>

#include <string>
#include <vector>
#include <map>
#include <algorithm>

struct Node{
    std::vector<int> ways;
    bool isLeaf = false;
    int str_num = -1;
    int link = -1;
    int flink = -1;
    int from = -1;
    char how = 0;
    std::vector<int> go;
};

Node createNode(int from, char how){
    Node vert;
    vert.ways = { -1, -1, -1, -1, -1 };
    vert.go = { -1, -1, -1, -1, -1 };
    vert.from = from;
    vert.how = how;
    return vert;
}

void addString(std::string& str, std::vector<Node>& bor, std::map<char, int>&
alphabet, int str_num){
    int borInd = 0;
    for (auto c : str){
        char cInd = alphabet[c];
        if (bor[borInd].ways[cInd] == -1){ bor.push_back(createNode(borInd, cInd));
bor[borInd].ways[cInd] = bor.size() - 1;}
        borInd = bor[borInd].ways[cInd];
    }
    bor[borInd].isLeaf = true;
    bor[borInd].str_num = str_num;
}

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

int getSuffLink(int v, std::vector<Node>& bor){
    if (bor[v].link == -1){
        if (v == 0 || bor[v].from == 0) bor[v].link = 0;
        else bor[v].link = go(getSuffLink(bor[v].from, bor), bor[v].how, bor);
    }return bor[v].link;
}

int go(int v, char c, std::vector<Node>& bor){
    if (bor[v].go[c] == -1){
        if (bor[v].ways[c] != -1) bor[v].go[c] = bor[v].ways[c];
```

```
        else{ if (v == 0) bor[v].go[c] = 0; else bor[v].go[c] = go(getSuffLink(v,
bor), c, bor); }
    } return bor[v].go[c];
}

int main(){
    std::map<char, int> alphabet;
    alphabet['A'] = 0;
    alphabet['C'] = 1;
    alphabet['G'] = 2;
    alphabet['T'] = 3;
    alphabet['N'] = 4;
    std::string text;
    size_t n;
    std::cin >> text >> n;
    std::vector<std::string> a(n);
    for (size_t i = 0; i < n; ++i) std::cin >> a[i];
    std::vector<Node> bor;
    bor.push_back(createNode(0, 0));
    for (size_t i = 0; i < a.size(); ++i) addString(a[i], bor, alphabet, i);
    int u = 0;
    for (int i = 0; i < text.length(); ++i){
        std::cout << "Элемент текста: " << text[i] << " Состояние: " << u <<
std::endl;
        u = go(u, alphabet[text[i]], bor);
        std::cout << "Новое состояние: " << u << std::endl;
        for (int v = u; v != 0; v = getSuffLink(v, bor)){
            std::cout << "\tВозврат к корню. Текущее состояние: " << v <<
std::endl;
            if (bor[v].isLeaf) std::cout << "\t\tЛист!!! Позиция в тексте: " << i -
a[bor[v].str_num].length() + 2 << " Номер образца: " << bor[v].str_num + 1 <<
std::endl;
        }
    }
}
```

ПРИЛОЖЕНИЕ В.

ИСХОДНЫЙ КОД

```
#INCLUDE <IOSTREAM>

#include <vector>
#include <map>
#include <fstream>

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

const int K = 5;
std::map<char, int> alphabet;

struct Node {
    int ways[K] = {-1, -1, -1, -1, -1};
    bool isLeaf = false;
    int numPrev = -1;
    char symbPrev = 0;
    int suffLink = -1;
    int go[K] = {-1, -1, -1, -1, -1};
    std::vector<int> leafPatternNumber;
};

void addString (const std::string& s, int num, std::vector<Node>& bor) {
    int v = 0;
    Node curr = Node();
    for (char i : s){ char c = alphabet[i];
        if (bor[v].ways[c] == -1) {
            curr.numPrev = v;
            curr.symbPrev = c;
            bor[v].ways[c] = bor.size();
            bor.push_back(curr);
        }v = bor[v].ways[c];}
    bor[v].isLeaf = true;
    bor[v].leafPatternNumber.push_back(num);
    std::cout<<"";
}

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

int getSuffLink (int v, std::vector<Node>& bor) {
    if (bor[v].suffLink == -1){
        if (v == 0 || bor[v].numPrev == 0) bor[v].suffLink = 0;
        else bor[v].suffLink = go(getSuffLink(bor[v].numPrev, bor),
bor[v].symbPrev, bor);
    }return bor[v].suffLink;
}

int go (int v, char c, std::vector<Node>& bor) {
```

```
    if (bor[v].go[c] == -1){
        if (bor[v].ways[c] != -1) bor[v].go[c] = bor[v].ways[c];
        else{ if(v == 0) bor[v].go[c] = 0; else bor[v].go[c] = go(getSuffLink(v,
bor), c, bor); }
    }return bor[v].go[c];
}

void multipleSearch(std::vector<std::pair<std::string, int>>& substrings,
std::vector<Node>& bor, std::string& text, std::string& str, std::ostream& fout) {
    for(int i = 0; i < substrings.size(); ++i) addString(substrings[i].first, i + 1
, bor);
    std::vector<int> templates_count(text.size());
    templates_count.insert(templates_count.begin(), 0);
    int count = 0;
    Node current = bor[0];
    std::ofstream output;
    fout << "Количество вершин: " << bor.size() << std::endl;
    for(int i = 0; i<text.size(); ++i) {
        char c = alphabet[text[i]];
        count = go(count, c, bor);
        fout<<"Номер вершины в боре: "<<count<<"\n\tПроход по суффиксным ссылкам:\
n";
        for (int v = count; v != 0; v = getSuffLink(v, bor)) {
            if(bor[v].isLeaf){
                for(int j = 0; j < bor[v].leafPatternNumber.size(); ++j) {
                    int pos = i - substrings[bor[v].leafPatternNumber[j] -
1].first.length() + 1;
                    fout<<"\tПозиция подстроки(A): "<< pos << "; ";
                    int t_index = pos - substrings[bor[v].leafPatternNumber[j] -
1].second + 1;
                    fout<<"Позиция в строке с маской(B): "<<
substrings[bor[v].leafPatternNumber[j] - 1].second + 1<<"; A-B =
"<<t_index<<std::endl;
                    if(t_index >= 0) {
                        templates_count[t_index] += 1;
                        if (templates_count[t_index] == substrings.size()) {
                            fout << "\tКоличество повторений в векторе шаблонов
равно количеству подстрок без масок. Шаблон найден.\n";
                            fout << "\tИндекс вхождения: " << t_index + 1 <<
std::endl <<"Пересечения с другими найденными образцами:\n";
                            for(int k = t_index + 1; k < t_index + str.size() - 1;
++k) {
                                if(templates_count[k - str.size() + 1] ==
substrings.size()) {
                                    fout << "\t" << 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";
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
int main() {
    alphabet['A'] = 0;
    alphabet['C'] = 1;
    alphabet['G'] = 2;
    alphabet['T'] = 3;
    alphabet['N'] = 4;
    char in, out;
    std::cout << "Для считывания/вывода через консоль введите - 'c', через файл -
'f' \n";
    std::cin >> in >> out;
    if((out != 'c' && out != 'f') || (in != 'c' && in != 'f')){ std::cout <<
"Неверный ввод\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("input.txt"); input >> text >> str >>
joker; }
    str += joker;
    std::vector<std::pair<std::string, int>> substrings;
    std::vector<Node> bor;
    Node root = Node();
    bor.push_back(root);
    std::pair<std::string,int> pairTmp;
    for(int i = 0; i < str.size(); i++){
        if(str[i] == joker){
            if(!pairTmp.first.empty()){
                pairTmp.second = i + 1 - pairTmp.first.length();
                substrings.push_back(pairTmp);
                pairTmp.first.clear();
            }continue;
        }pairTmp.first += str[i];}
    if(!pairTmp.first.empty()){ pairTmp.second = str.size() -
pairTmp.first.length(); substrings.push_back(pairTmp); }
    if(out=='c') multipleSearch(substrings, bor, text, str, std::cout);
    else{ std::ofstream fout("output.txt"); multipleSearch(substrings, bor, text,
str, std::cout); }
    return 0;
}
```