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

Студентка гр. 7383 _____ Прокопенко Н.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Ознакомиться с алгоритмом Ахо-Корасика для эффективного поиска всех вхождений всех строк-образцов в заданную строку.

.

Реализация задачи

Для решения поставленной задачи был написан класс `Aho_Karasik` и структура `bohr_vertex`. Структура используется для реализации бора. 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. В этой структуре мы используем поля `int obraz` – номер строки-образца, `bool flag` – бит, показывающий является ли вершина исходной строкой. `map<char,int> edge` – вершины, в которые мы можем пойти из данной, `map<char,int> auto_move` – запоминает переходы автомата. Переход выполняется по двум параметрам — текущей вершине `m_par` и символу `m_symb`. по которому нам надо сдвинуться из этой вершины. Необходимо найти вершину u, которая обозначает наидлиннейшую строку, состоящую из суффикса строки `m_par` (возможно нулевого) + символа `m_symb`. Если такого в боре нет, то идем в корень, `int suff_link` – суффиксная ссылка, `int par` – индекс вершины родителя, `char symb` – символ на ребре от родителя к этой вершине. В классе `Aho_Karasik` хранятся `vector<bohr_vertex> bohr` – вектор, для хранения вершин, `vector <string> patterns` – вектор для хранения строк-шаблонов, `int counter` – счетчик вершин. Также реализованы некоторые методы: `void add_string_to_bohr(string& s, int str)` – метод, создающий бор и добавляющий строки в `patterns`, `void find_all_pos(string s)` – метод, который выполняет автоматные переходы и выводит ответ на экран, `int get_suff_link(int v)` – возвращает суффиксальную ссылку для данной 4

вершины, `int get_auto_move(int v, char symbol)` – метод, который выполняет автоматные переходы, `void check(int v, int i)` – метод, который осуществляет хождение по хорошим суффиксальнм ссылкам из текущей позиции, учитывая, что эта позиция оканчивается на символ i.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Так же было проведено исследование алгоритма. Структура данных map из STL реализована красно-черным деревом, а время обращения к его элементам пропорционально логарифму числа элементов. Следовательно вычислительная сложность O((H + n)logk + c), где H — длина текста, в котором производится поиск, n — общая длина всех слов в словаре, k — размер алфавита, c — общая длина всех совпадений. Сложность по памяти O(H + n), т.к. память выделяется для вершин шаблонов и для хранения текста.

Выводы

В ходе данной лабораторной работы был реализован алгоритм Ахо-Корасика на языке C++. Данный алгоритм производит точный поиск набора образцов в строке. Были изучены новые структуры данных и понятия, такие как бор, суффиксальные ссылки и.т.д.. Код программы представлен в приложении А.

ПРИЛОЖЕНИЕ А.
КОД ПРОГРАММЫ

Lab5_1.cpp

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;
struct bohr_vertex {
    bool flag;
    int obraz;//номер образца
    int suff_link;//суффиксная ссылка
    int par;//вершина-отец в дереве
    char symb; //символ на ребре от par к этой вершине
    map<char,int> edge;//номер вершины, в которую мы придем по символу с
номером i в алфавите
    map<char,int> auto_move;//auto_move - запоминание перехода автомата
    bohr_vertex(int m_par, char m_symb): par(m_par), symb(m_symb) {
        flag = false;
        obraz = 0;
        suff_link = -1;//изначально - суф. ссылки нет
    }
};
class Aho_Karasik{
    vector<bohr_vertex> bohr;
    vector <string> patterns;
    int counter;//счетчик узлов бора
public:
    Aho_Karasik(){//создание корня бора
        counter = 1;
        bohr.push_back(bohr_vertex(0, 0));
    }
    void  add_string_to_bohr(string&  s,  int  str){//добавление  строки-
образца в бор
        int n = 0;//начинаем с корня
        patterns.push_back(s);
        for(int i = 0; i < s.length(); i++){
            if(bohr[n].edge.find(s[i])  ==  bohr[n].edge.end()){//если  от
вершины нет путей в искомую
                bohr.push_back(bohr_vertex(n, s[i]));
                bohr[n].edge[s[i]] = counter++;
            }
            n = bohr[n].edge[s[i]];
        }
        bohr[n].flag = true;//n - конечная вершина
        bohr[n].obraz = patterns.size();
```

```
    }
    int get_suff_link(int v){//возвращает индекс суффиксной ссылки
        if(bohr[v].suff_link == -1){//если еще не считали
            if (v == 0 || bohr[v].par == 0) //если v - корень или предок
v - корень
                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 symbol){//переходы автомата
        if(bohr[v].auto_move.find(symbol) == bohr[v].auto_move.end())
            if(bohr[v].edge.find(symbol) != bohr[v].edge.end())
                bohr[v].auto_move[symbol] = bohr[v].edge[symbol];
            else
                if (v == 0)//если v - корень
                    bohr[v].auto_move[symbol] = 0;
                else
                    bohr[v].auto_move[symbol]                            =
get_auto_move(get_suff_link(v), symbol);
        return bohr[v].auto_move[symbol];
    }
    void  check(int  v,  int  i)  {   //хождение  по  хорошим  суффиксальнм
ссылкам из текущей позиции, учитывая, что эта позиция оканчивается на
символ i
        for (int u = v; u != 0; u = get_suff_link(u))
            if (bohr[u].flag)
                cout<< i - patterns[bohr[u].obraz - 1].size() + 2 << " "
<< bohr[u].obraz << endl;
    }
    void find_all_pos(string s) {
        int u = 0;
        for (int i = 0; i < s.length(); i++) {
            u = get_auto_move(u, s[i]);
            check(u, i);
        }
    }
};
int main(){
    Aho_Karasik ah;
    string text, pattern;
    int num;
    cin >> text >> num;
    for(int i = 0; i < num; i++){
```

```
        cin >> pattern;
        ah.add_string_to_bohr(pattern, i + 1);
    }
    ah.find_all_pos(text);
    return 0;
}
```

Lab5_2.cpp

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;
struct bohr_vertex {
    bool flag;
    int obraz;//номер образца
    int suff_link;//суффиксная ссылка
    int par;//вершина-отец в дереве
    char symb; //символ на ребре от par к этой вершине
    map<char,int> edge;//номер вершины, в которую мы придем по символу с
номером i в алфавите
    map<char,int> auto_move;//auto_move - запоминание перехода автомата
    bohr_vertex(int m_par, char m_symb): par(m_par), symb(m_symb) {
        flag = false;
        obraz = 0;
        suff_link = -1;//изначально - суф. ссылки нет
    }
};
class Aho_Karasik{
    vector<bohr_vertex> bohr;
    vector <string> patterns;
    int counter;//счетчик узлов бора
    char joker;
public:
    Aho_Karasik(char _joker, string& s): joker(_joker){//создание корня
бора
        counter = 1;
        bohr.push_back(bohr_vertex(0, 0));
        int n = 0;//начинаем с корня
        patterns.push_back(s);
        for(int i = 0; i < s.length(); i++){//добавление строки-образца
в бор
            if(bohr[n].edge.find(s[i]) == bohr[n].edge.end()){//если от
вершины нет путей в искомую
                bohr.push_back(bohr_vertex(n, s[i]));
                bohr[n].edge[s[i]] = counter++;
            }
```

```
            n = bohr[n].edge[s[i]];
        }
        bohr[n].flag = true;//n - конечная вершина
        bohr[n].obraz = patterns.size();
    }
    int get_suff_link(int v){//возвращает индекс суффиксной ссылки
        if(bohr[v].suff_link == -1){//если еще не считали
            if (v == 0 || bohr[v].par == 0) //если v - корень или предок
v - корень
                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 symbol){//переходы автомата
        if(bohr[v].auto_move.find(symbol) == bohr[v].auto_move.end())
            if(bohr[v].edge.find(symbol) != bohr[v].edge.end())
                bohr[v].auto_move[symbol] = bohr[v].edge[symbol];
            else if(bohr[v].edge.find(joker) != bohr[v].edge.end())
                    bohr[v].auto_move[symbol] = bohr[v].edge[joker];
            else
                if (v == 0)//если v - корень
                    bohr[v].auto_move[symbol] = 0;
                else
                    bohr[v].auto_move[symbol]                        =
get_auto_move(get_suff_link(v), symbol);
        return bohr[v].auto_move[symbol];
    }
    void check(int v, int i) {   //хождение по хорошим суффиксальнм
ссылкам из текущей позиции, учитывая, что эта позиция оканчивается на
симво i
        for (int u = v; u != 0; u = get_suff_link(u))
            if (bohr[u].flag)
                cout<< i - patterns[bohr[u].obraz - 1].size() + 2 <<
endl;
    }
    void find_all_pos(string s) {
        int u = 0;
        for (int i = 0; i < s.length(); i++) {
            u = get_auto_move(u, s[i]);
            check(u, i);
        }
    }
};
```

```
int main(){
    string text, pattern;
    char joker;
    cin >> text >> pattern >> joker;
    Aho_Karasik ah(joker, pattern);
    ah.find_all_pos(text);
    return 0;
}
```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены в табл. 1.

Входные данные	Выходные данные
ACT A$ $	1
ACATC A$ $	1 3
ACATC C$$ $	2