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. 7383 _____ Тян Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: исследовать и реализовать задачу поиска набора образцов в тексте с помощью алгоритма Ахо-Корасик.

Формулировка задачи: необходимо разработать программу, которая реализует алгоритм Ахо-Корасик и с его помощью для заданных шаблонов $P = \{p_1, \ldots, p_n\}$ $(1 \leq |p_i| \leq 75)$ и текста T $(1 \geq |T| \leq 100000)$ найти все вхождения образцов из P в T.

Также следует разработать программу для решения следующей задачи: 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 шаблон входит хотя бы один символ не джокер.

Входные данные для первого задания:

Первая строка – T.

Вторая строка – число n $(1 \leq n \leq 3000)$, каждая из n строк содержит шаблон из набора P.

Входные данные для второго задания:

Первая строка – T $(1 \leq |T| \leq 100000)$.

Вторая строка – P $(1 \leq |P| \leq 40)$.

Третья строка – символ джокера.

Программа выводит все вхождения образцов из P в T. Каждое вхождение образца в текст представляется в виде двух чисел – i p, где i – позиция в тексте, с которой начинается вхождение образца с номером p. Строки выхода отсортированы по возрастанию сначала номера позиции, затем шаблона.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В данной работе используются главная функция (int main()) и класс (bor), содержащий конструктор по умолчанию и методы (void init_b(int i, string p), void make_suf(int i), void cout_prev(int index, int str_index), void searching(string t)), а также структура vertex.

Свойства класса bor:

- bohr –бор, представленный в виде массива структур vertex;
- pats – массив шаблонов.

Параметры, передаваемые в метод void init_b(int i, string p):

- i – номер шаблона;
- p – i-ый шаблон.

Параметры, передаваемые в метод void make_suf(int i):

- i – номер вершины, для которой необходимо построить суффиксальную ссылку.

Параметры, передаваемые в метод void cout_prev(int index, int str_index):

- index – вершина, начиная с которой проверяются все суффиксальные ссылки;
- str_index – индекс символа в тексте, который будет концом суффикса для любого префикса, найденного по суффиксальной ссылке.

Параметры, передаваемые в метод void searching(string t):

- t – заданный текст.

В функции main() считывается текст T и число n, характеризующее количество шаблонов. Создается объект класса bor. Начинается считывание шаблонов и вызов метода void init_b(int i, string p) ранее созданного объекта. В методе void init_b(int i, string p) каждый символ из шаблона p проходит проверку на наличие данного символа в боре как продолжение какого-либо префикса, если в боре не нашелся ни один такой префикс, то создается вершина как продолжение данного префикса и добавляется в бор, представленный в виде массива. Как только все шаблоны были считаны и их

символы добавлены в бор, вызывается метод void searching(string t), и начинается поиск в тексте данных шаблонов. В методе void searching(string t) проверяется каждый символа текста, если совпадение не найдено ни с одним префиксом для данного бора, то символ пропускается, и проверку проходит следующий. Если символ совпал с продолжением данного префикса, то проверяется, является ли этот символ концом данного шаблона, если это так, то выводится номер позиции, с которой начинается данный шаблон и номер шаблона. Даже если символ не является концом шаблона, вызывается метод void cout_prev(int index, int str_index), который смотрим все предыдущие шаблоны, которые могли были войти в данный префикс с помощью суффиксальных ссылок. Если символ не совпал, то происходит переход по суффиксальной ссылке пока символ не совпадет или пока не дойдем до корня.

Для более понятной работы алгоритма рассмотрим пример работы программы для примера:

baaacbaac

3

a

aa

baac

Построим бор для данного набора шаблонов. При создании объекта

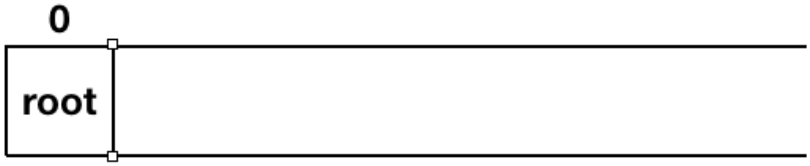

Рисунок 1 — Инициализация бора, бор в виде массива

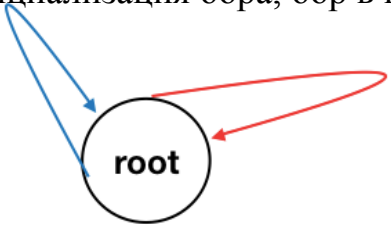

Рисунок 2 — Инициализация бора, бор в виде графа

класса bor, сразу же создается корень данного бора – вершина, пустая,

суффиксальный ссылка которой указывает сама на себя и родителем которой является она сама, см. рис 1, 2. Далее идет шаблон и символ «a». Так как для

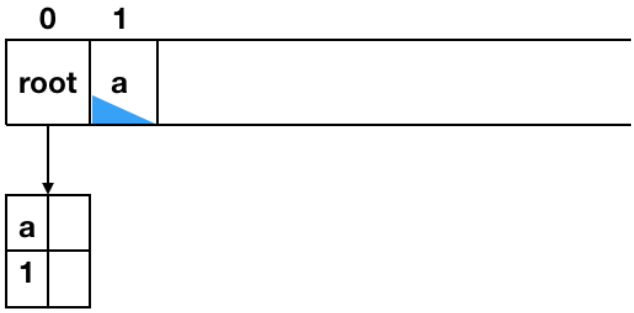

Рисунок 3 — Результат добавления первого шаблона, бор в виде массива

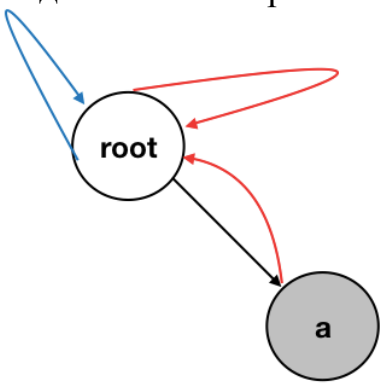

Рисунок 4 — Результат добавления первого шаблона, бор в виде графа

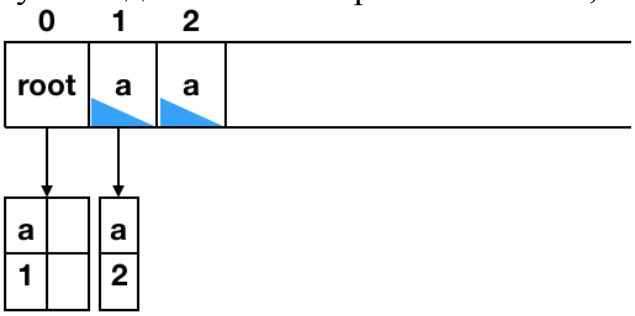

Рисунок 5 — Результат добавления второго шаблона, бор в виде массива

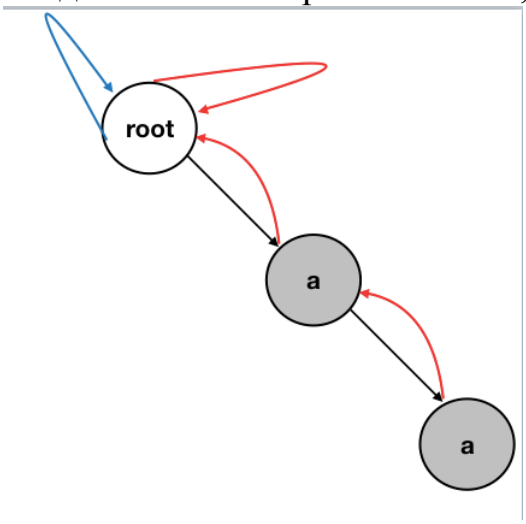

Рисунок 6 — Результат добавления второго шаблона, бор в виде графа

данного символа не существует префикса, содержащегося в боре, то

создается вершина, содержащая данный символ, родителем которой является

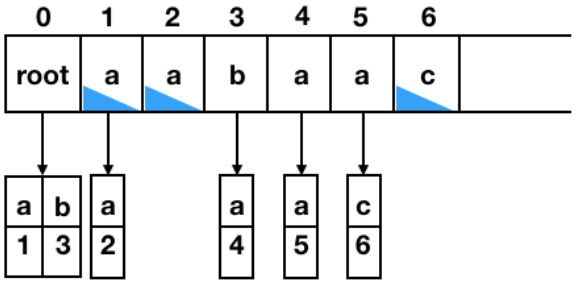

Рисунок 7 — Результат добавления третьего шаблона, бор в виде массива

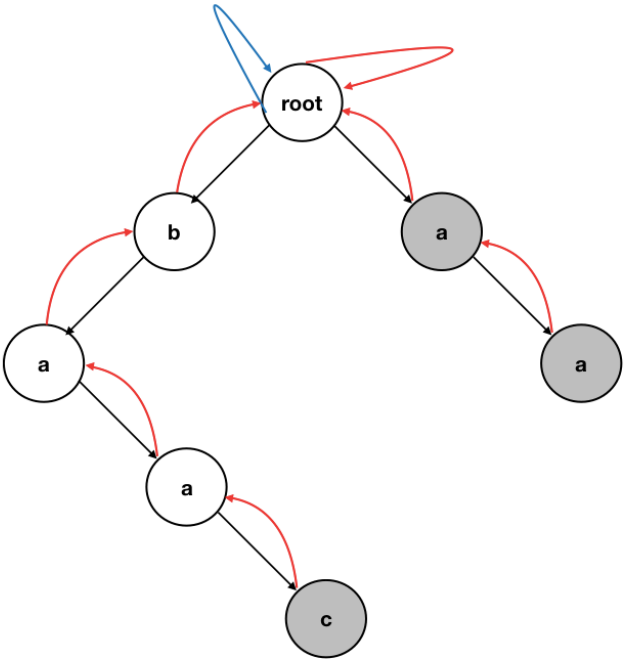

Рисунок 8 — Результат добавления третьего шаблона, бор в виде графа

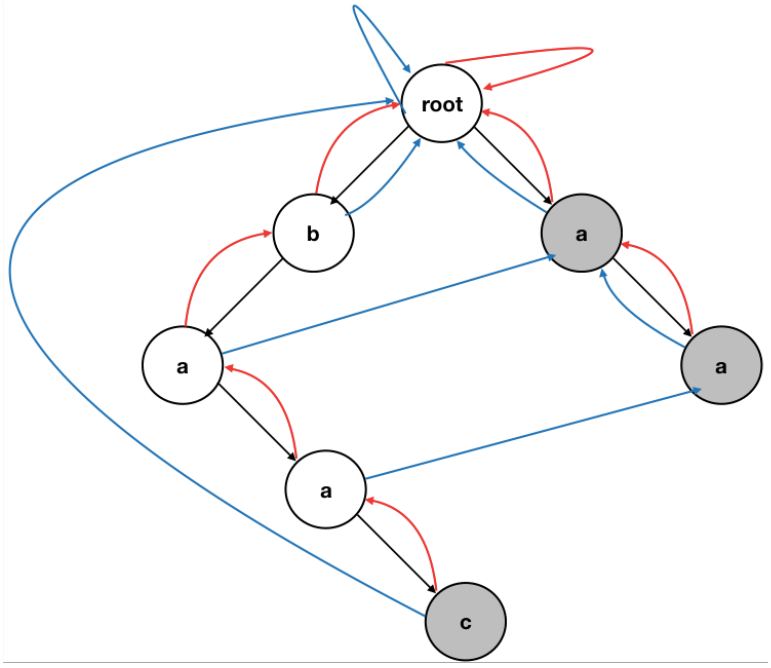

Рисунок 9 — Результат добавления суффиксальных ссылок, бор в виде графа

корень. Иллюстрация добавления первого шаблон в бор приведена на рис. 3, 4. Следующий шаблон – «aa». Для первого символа уже существует ребро. Значит просто переходит по этому ребру в смежную вершину. Для второго символа не существует ребра, тогда создаем его и добавляем данную вершину в бор. Иллюстрация добавления второго шаблона в бор приведена на рис. 5, 6. Для третьего шаблона будет создан целый новый путь в боре, иллюстрация которого приведена на рис. 7, 8.

Далее строятся суффиксальные ссылки по правилу: чтобы построить суффиксальную ссылку, поднимаемся до родителя и переходим по его ссылке, далее смотрим, есть ли ребро с таким же символом, если есть, то для данного символа строим суффиксальную ссылку. Так на рис. 9, чтобы построить суффиксальную ссылку для вершины «a», ребенком, которой является вершина «c», нужно подняться до родителя, т.е. предыдущей вершины и перейти по суффиксальной ссылке. Так перешли в вершину «a», которая исходит из корня, так как у нее есть ребенок с тем же символом, то строим суффиксальную ссылку для исходной вершины на ребенка с тем же символом.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе G++ в OS Linux Ubuntu 12.04. Программа может быть скомпилирована с помощью команды:

g++ <имя файла>.cpp -std=c++11

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно увидеть, что тестовые случаи не выявили некорректной работы программы, что говорит о том, что по результатам тестирования было показано, что поставленная задача была выполнена.

4. ИССЛЕДОВАНИЕ

Компиляция была произведена в компиляторе в g++ Version 4.2.1. В других ОС и компиляторах тестирование не проводилось.

В начале работы программа строит бор, далее память не расходуется. Тогда можно оценить алгоритм по памяти как – O($n \cdot \Sigma$), где n – общая длина всех шаблонов, а Σ – размер алфавита. Также производится поиск всевозможных вхождений шаблонов в текст – O($n \cdot \Sigma + |T| + k$), где $|T|$ – длина текста, n – длина совпадений.

5. ВЫВОД

В ходе выполнения лабораторной работы была решена задача поиска набора шаблонов в тексте с помощью алгоритма Ахо-Корасик на языке C++, и исследован алгоритм Ахо-Корасик.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

№	Ввод	Вывод
1	aaaa 3 a aa aaa	1 1 1 2 2 1 1 3 2 2 3 1 2 3 3 2 4 1
2	ccca 2 cc cca	1 1 2 1 2 2
3	ccca 2 mk ccq	
4	baaacbaac 3 a aa baac	2 1 2 2 3 1 3 2 4 1 7 1 7 2 8 1 6 3

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp:

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

using namespace std;

struct vertex
{
    int num_pat = 0;
    vector <int> arr;
    int suff_link = -1;
    char symbol;
    int parent = -1;
};

class bor
{
public:
    void make_suf(int i){
        int tmp = bohr[i].parent;
        int temp;
        while(true){
            if(tmp == 0){
                bohr[i].suff_link = 0;
                break;
            }
            int k = 0;
            if(bohr[tmp].suff_link != -1){
                for(k = 0; k < bohr[bohr[tmp].suff_link].arr.size(); k++){
                    if(bohr[bohr[bohr[tmp].suff_link].arr[k]].symbol ==
bohr[i].symbol){
                        bohr[i].suff_link = bohr[bohr[tmp].suff_link].arr[k];
                        break;
                    }
                }
            }else{
                temp = bohr[tmp].parent;
            }
            if(bohr[tmp].suff_link != -1 && k ==
bohr[bohr[tmp].suff_link].arr.size()){
                temp = bohr[tmp].suff_link;
            }else
                break;
            tmp = temp;
        }
    }
    bor(){
        vertex root;
        root.suff_link = 0;
        root.parent = 0;
        bohr.push_back(root);
    }
    void init_b(int i, string p)
    {
```

```
            pats.push_back(p);
            int it;
            int curr_v = 0;
            for(int j = 0; j < p.length(); j++)
            {
                if(bohr[curr_v].arr.empty())
                {
                    it = -1;
                }else{
                    int k;
                    for(k = 0; k < bohr[curr_v].arr.size(); k++){
                        if(bohr[bohr[curr_v].arr[k]].symbol == p[j]){
                            it = bohr[curr_v].arr[k];
                            break;
                        }
                    }
                    if(k == bohr[curr_v].arr.size()){
                        it = -1;
                    }
                }
                if(it == 0 || it == -1)
                {
                    struct vertex node;
                    node.symbol = p[j];
                    node.parent = curr_v;
                    bohr.push_back(node);
                    bohr[curr_v].arr.push_back(bohr.size() - 1);
                    it = bohr.size() - 1;
                }
                curr_v = it;
                if(j == p.size() - 1){
                    bohr[curr_v].num_pat = i;
                }
            }
    }
    void cout_prev(int index, int str_index){
        while(index != 0){
            if(bohr[index].suff_link == -1)
                make_suf(index);
            if(bohr[index].num_pat != 0){
                  cout  << str_index + 2 - pats[bohr[index].num_pat - 1].length() <<
' ' << bohr[index].num_pat << endl;
            }
            index = bohr[index].suff_link;
        }
    }
    void searching(string t)
    {
        int cur_v = 0;
        for(int i = 0; i < t.length(); i++){
            int j = -1;
            int tmp = cur_v;
            if(bohr[cur_v].suff_link == -1){
                make_suf(cur_v);
            }
            for(j = 0; j < bohr[cur_v].arr.size(); j++)
            {
                if(bohr[bohr[cur_v].arr[j]].suff_link == -1){
                    make_suf(bohr[cur_v].arr[j]);
```

```
                }
                if(bohr[bohr[cur_v].arr[j]].symbol == t[i]){
                    if(bohr[bohr[cur_v].arr[j]].num_pat != 0){
                         cout << i + 2 - pats[bohr[bohr[cur_v].arr[j]].num_pat -
1].length() << ' ' << bohr[bohr[cur_v].arr[j]].num_pat << endl;
                    }
                    cout_prev(bohr[bohr[cur_v].arr[j]].suff_link, i);
                    tmp = bohr[cur_v].arr[j];
                    break;
                }
            }
            if(j == bohr[cur_v].arr.size()){
                if(cur_v != 0){
                    if(bohr[cur_v].suff_link != -1)
                        tmp = bohr[cur_v].suff_link;
                    else
                        tmp = bohr[cur_v].parent;
                    i--;
                }
            }
            cur_v = tmp;
        }
    }
private:
    vector<struct vertex> bohr;
    vector<string> pats;
};

int main()
{
    string t;
    cin >> t;
    int n;
    cin >> n;
    bor B;
    string p;
    for(int i = 1; i <= n; i++){
        cin >> p;
        B.init_b(i,p);
    }
    B.searching(t);
    return 0;
}
```

joker.cpp:

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

using namespace std;

struct vertex
{
    int num_pat = 0;
    vector <int> arr;
    int suff_link = -1;
    char symbol;
```

```
    int parent = -1;
};

class bor
{
public:
    void make_suf(int num){
        int i = num;
        int tmp = bohr[i].parent;
        while(true){
            if(tmp == 0){
                bohr[i].suff_link = 0;
                break;
            }
            int k = 0;
            for(k = 0; k < bohr[bohr[tmp].suff_link].arr.size(); k++){
                if(bohr[bohr[bohr[tmp].suff_link].arr[k]].symbol == bohr[i].symbol){
                    bohr[i].suff_link = bohr[bohr[tmp].suff_link].arr[k];
                    break;
                }
            }
            if(k == bohr[bohr[tmp].suff_link].arr.size()){
                tmp = bohr[tmp].parent;
            }else
                break;
        }
    }
    bor(string p)
    {
        struct vertex root;
        root.suff_link = 0;
        root.parent = 0;
        bohr.push_back(root);
        pats.push_back(p);
        int it;
        int curr_v = 0;
        for(int j = 0; j < p.length(); j++)
        {
            if(bohr[curr_v].arr.empty())
            {
                it = 0;
            }else{
                int k;
                for(k = 0; k < bohr[curr_v].arr.size(); k++){
                    if(bohr[bohr[curr_v].arr[k]].symbol == p[j]){
                        it = bohr[curr_v].arr[k];
                        break;
                    }
                }
                if(k == bohr[curr_v].arr.size() &&  bohr[bohr[curr_v].arr[k -
1]].symbol != p[j]){
                    it = -1;
                }
            }
            if(it == 0 || it == -1)
            {
                struct vertex node;
                node.symbol = p[j];
                node.parent = curr_v;
```

```
                bohr.push_back(node);
                bohr[curr_v].arr.push_back(bohr.size() - 1);
                it = bohr.size() - 1;
            }
            curr_v = it;
            if(j == p.size() - 1){
                bohr[curr_v].num_pat = 1;
            }
        }
        for(int i = 1; i < bohr.size(); i++){
            make_suf(i);
        }
    }
    void cout_prev(int index, int str_index){
        while(index != 0){
            if(bohr[index].num_pat != 0){
                cout  << str_index + 2 - pats[bohr[index].num_pat - 1].length() <<
endl;
            }
            index = bohr[index].suff_link;
        }
    }
    void searching(string t, char joker)
    {
        int cur_v = 0;
        for(int i = 0; i < t.length(); i++){
            int j = -1;
            int tmp = cur_v;
            for(j = 0; j < bohr[cur_v].arr.size(); j++)
            {
                if(bohr[bohr[cur_v].arr[j]].symbol == t[i]){
                    if(bohr[bohr[cur_v].arr[j]].num_pat != 0){
                        cout << i + 2 - pats[bohr[bohr[cur_v].arr[j]].num_pat -
1].length() << endl;
                    }
                    cout_prev(bohr[bohr[cur_v].arr[j]].suff_link, i);
                    tmp = bohr[cur_v].arr[j];
                    break;
                }else if(bohr[bohr[cur_v].arr[j]].symbol == joker)
                {
                    if(bohr[bohr[cur_v].arr[j]].num_pat != 0){
                        cout << i + 2 - pats[bohr[bohr[cur_v].arr[j]].num_pat -
1].length() << endl;
                    }
                    cout_prev(bohr[bohr[cur_v].arr[j]].suff_link, i);
                    tmp = bohr[cur_v].arr[j];
                    break;
                }
            }
            if(j == bohr[cur_v].arr.size()){
                if(cur_v != 0){
                    tmp = bohr[cur_v].suff_link;
                    i--;
                }
            }else
                if(bohr[cur_v].arr.empty()){
                    tmp = bohr[cur_v].suff_link;
                    i--;
                }
```

```
            cur_v = tmp;
        }
    }
private:
    vector<struct vertex> bohr;
    vector<string> pats;
};

int main()
{
    string t,p;
    char j;
    cin >> t >> p >> j;
    bor B(p);
    B.searching(t, j);
    return 0;
}
```