

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-КАРАСИК.

Студент гр. 7304

Моторин Е.В

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2019

Цель работы:

Исследовать алгоритм Ахо-Корасик, который реализует поиск множества подстрок из словаря в данной строке.

Задача:

-
- 1) Разработайте программу, решающую задачу точного поиска набора образцов.
 - 2) Используя реализацию точного множественного поиска, решить задачу поиска для одного образца с джокером.

Основные теоретические положения:

Бор (англ. trie, луч, нагруженное дерево) — структура данных для хранения набора строк, представляющая из себя подвешенное дерево с символами на рёбрах. Строки получаются последовательной записью всех символов, хранящихся на рёбрах между корнем бора и терминальной вершиной. Размер бора линейно зависит от суммы длин всех строк, а поиск в бору занимает время, пропорциональное длине образца.

Детерминированный конечный автомат (ДКА) (англ. deterministic finite automaton (DFA)) — набор из пяти элементов $\langle \Sigma, Q, s \in Q, T \subset Q, \delta: Q \times \Sigma \rightarrow Q \rangle$, где Σ — алфавит (англ. alphabet), Q — множество состояний (англ. finite set of states), s — начальное (стартовое) состояние (англ. start state), T — множество допускающих состояний (англ. set of accept states), δ — функция переходов (англ. transition function).

Ход работы:

1. Реализован алгоритм Ахо-Карасик

- а. Была реализована структура вершины в боре. Next_vrtx соседние вершины, pat_num — номер строки, suff_link — суффиксная ссылка, auto_move — массив переходов автомата, par — предок вершины, flag — флаг, который показывает удовлетворяет ли вершина шаблону или нет, number — порядковый номер введенной строки.

```
struct bohr_vrtx {
    int next_vrtx[k], pat_num, suff_link, auto_move[k], par, suff_flink;
    bool flag;
    int number;
    char symb;
```

```
};
bohr_vrtx make_bohr_vrtx(int p,char c){
    bohr_vrtx v;
    memset(v.next_vrtx, 255, sizeof(v.next_vrtx));
    memset(v.auto_move, 255, sizeof(v.auto_move));
    v.flag=false;
    v.suff_link=-1;
    v.par=p;
    v.symb=c;
    v.suff_flink=-1;
    return v;
}
```

- b. Была реализована функция, добавляющая строку в бор. Сначала мы в нулевой корневой вершине. Далее идем по доступным вершинам, если нельзя то создаем вершину и переход и переходим по ней.

```
void add_string_to_bohr(string& s,int n){
    int num=0;
    for (int i=0; i<s.length(); i++){
        char ch = sym.find(s[i]);
        if (bohr[num].next_vrtx[ch]==-1){
            bohr.push_back(make_bohr_vrtx(num,ch));
            bohr[num].next_vrtx[ch]=bohr.size()-1;
        }
        num=bohr[num].next_vrtx[ch];
    }
    bohr[num].flag=true;
    bohr[num].number = n;
    pattern.push_back(s);
    bohr[num].pat_num=pattern.size()-1;
}
```

- c. Были реализованы 2 функции, которые строят по данному бору конечный детерминированный автомат, который строится по ходу работы алгоритма. Получаем суффиксальную ссылку для данной вершины. Если она не определена ($=-1$), то сначала проверяем не корень ли это или ее предок корень, тогда суфф. Ссылка будет 0, иначе переходим на предка и пытаемся пройти по данному символу, если нельзя повторяем. В функции `get_auto_move` осуществляется переход, который выдает следующее состояние.

```

int get_auto_move(int v, char ch);
int get_suff_link(int v){
    if (bohr[v].suff_link==-1)
        if (v==0||bohr[v].par==0)
            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 ch){
    if (bohr[v].auto_move[ch]==-1)
        if (bohr[v].next_vrtx[ch]!=-1)
            bohr[v].auto_move[ch]=bohr[v].next_vrtx[ch];
        else
            if (v==0)
                bohr[v].auto_move[ch]=0;
            else
                bohr[v].auto_move[ch]=get_auto_move(get_suff_link(v), ch);
    return bohr[v].auto_move[ch];
}
int get_suff_flink(int v){
    if (bohr[v].suff_flink==-1){
        int u=get_suff_link(v);
        if (u==0)
            bohr[v].suff_flink=0;
        else
            bohr[v].suff_flink=(bohr[u].flag)?u:get_suff_flink(u);
    }
    return bohr[v].suff_flink;
}

```

2. Реализована задача множественного поиска

- a. Задача имеет некоторые отличия от предыдущей.
- b. Паттерн-строка разделяется на строки джокерами, запоминаем их индексы.
- c. Создаем массив нулей размера текста
- d. Для каждого вхождения паттерна инкрементируем $i - j + 1$ позицию в массиве, где i – индекс вхождения, j – позиция данного паттерна в исходно паттерне.

- е. В тех позициях массива, где значение совпадает с количеством паттернов, присутствует совпадение с исходным.

Результат:

1)

СССА

1

СС

1 1

2 1

admin-2:lr5 admin\$ █

2)

АСТ

А\$

\$

1

admin-2:lr5 admin\$ █

Вывод:

Таким образом, в ходе данной лабораторной работы был реализован алгоритм Ахо-Карасика на языке с++. Данный алгоритм точный поиск набора образцов в строке. Используются такие понятия, как бор, конечный автомат, суффиксальные ссылки. Также алгоритм был использован для поиска в строке паттерна, содержащего джокер.

В ходе данной работы столкнулся с проблемой построения конечного автомата и переходам по суффиксальным ссылкам, а также с проблемой реализация поиска строки с джокером.