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 _____ Сычевский Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Содержание

Цель работы

Цель работы: познакомиться с алгоритмом поиска подстрок в строке Ахо-Корасик, создать программу, осуществляющую поиск набора подстрок в строке с помощью алгоритма Ахо-Корасик, а также программу, осуществляющую поиск подстроки с масками в строке.

Формулировка задачи: Разработайте программу, решающую задачу точного поиска набора образцов. Первая строка содержит текст (T,$1 \leq |T| \leq 100000$). Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P=\{p_1,\ldots,p_n\} 1 \leq |p_i| \leq 75$. Все строки содержат символы из алфавита {A,C,G,T,N}.

Реализация задачи

Программу было решено писать на языке программирования C++.

Для реализации поставленной задачи был создан класс Bohr:

```
class Bohr{
    vector<bohr_vrtx> bohr;
    vector<string> pattern;
};
```

Также была создана вспомогательная структура bohr_vrtx:

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

Функция add_string_to_bohr()создает новые элементы бора. Функции get_suff_link() и get_suff_flink() расставляют суффиксные и финишные ссылки. Функция find_all_pos() осуществляет поиск вхождений шаблонов в строке. Функция check() выводит пары: конец вхождения шаблона и его номер.

Исходный код программы представлен в приложении Б.

Исследование алгоритма

Сложность алгоритма Ахо-Корасик O(n|A|+H+k), где H – длина текста, в котором производится поиск, n – общая длина всех слов в словаре, |A| – размер алфавита, k – общая длина всех совпадений.

Тестирование

1. Процесс тестирования

Программа собрана в операционной системе Windows 10. В других ОС и компиляторах тестирование не проводилось.

2. Результаты тестирования

В результате тестирования программы ошибок выявлено не было. Тестовые случаи представлены в приложении А.

Вывод

В ходе выполнения данной работы был изучен метод поиска набора подстрок в строке с помощью алгоритма Ахо-Корасик. Была написана программа, применяющая алгоритм Ахо-Корасик для поиска набора подстрок в строке, а также осуществляющая поиск поиск подстроки с масками в строке. Сложность алгоритма Ахо-Корасик составляет O(n|A|+H+k).

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

Входные данные	Результат
CCCA 1 CC	1 1 2 1
ACT A? ?	1

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

```
#include <iostream>
#include <cstring>
#include <vector>

using namespace std;
const int k = 60;

struct bohr_vrtx{
    int next_vrtx[k];
    int pat_num;
    int suff_link;
    int auto_move[k];
    int par;
    int suff_flink;
    bool flag;
    char symb;
    int count_arc;
};

class Bohr{
    vector<bohr_vrtx> bohr;
    vector<string> pattern;

    bohr_vrtx make_bohr_vrtx(int p, char c){
        bohr_vrtx v;
        memset(v.next_vrtx, -1, sizeof(v.next_vrtx));
        memset(v.auto_move, -1, sizeof(v.auto_move));
        v.flag = false;
        v.suff_link = -1;
        v.par = p;
        v.symb = c;
        v.suff_flink = -1;
        v.count_arc = 0;
        return v;
    }


public:
    Bohr(){
        bohr.push_back(make_bohr_vrtx(0, '$'));
    }

    void add_string_to_bohr(const string &s){
        int num = 0;
        for (int i = 0; i < s.length(); i++){
            int ch = s[i] - 'A';
            if (bohr[num].next_vrtx[ch] == -1){
                bohr[num].count_arc++;
                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;
        pattern.push_back(s);
        bohr[num].pat_num = pattern.size() - 1;
    }
private:

    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;
    }

    void check(int v, int i){
        for (int u = v; u != 0; u = get_suff_flink(u)){
            if (bohr[u].flag){
                cout << i - pattern[bohr[u].pat_num].length() + 1 << " "
<< bohr[u].pat_num + 1 << endl;
            }
        }
```

```
    }
public:
    void find_all_pos(const string &s){
        int u = 0;
        for (int i = 0; i < s.length(); i++){
            u = get_auto_move(u, s[i] - 'A');
            check(u, i + 1);
        }
    }
};

int main(){
    string T, pattern;
    int n;
    cin >> T >> n;
    Bohr bohr;
    for (int i = 0; i < n; i++){
        cin >> pattern;
        bohr.add_string_to_bohr(pattern);
    }
    bohr.find_all_pos(T);
}
```