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

Студент гр. 7383 _____ Левкович Д.В.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2019

Содержание

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 AK и структура 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 конкатенируя буквы из ребер в порядке обхода, мы получим строку, соответствующую этой вершине. Переход из состояний осуществляется по 2 параметрам — текущей вершине v и символу ch. по котороромy нам надо сдвинуться из этой вершины. Поконкретнее, необходимо найти вершину u, которая обозначает наидлиннейшую строку, состоящую из суффикса строки v (возможно нулевого) + символа ch. Если такого в боре нет, то идем в корень. Таким образом строится конечный автомат. Далее подается текст для поиска всех шаблонов.

Тестирование

Программа собрана в операционной системе Windows 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.

Сложность алгоритма

Существующий вариант алгоритм проходит циклом по длине s (N=s.length()), откуда его уже можно оценить как O(N*O(check)), но так как check прыгает только по заведомо помеченным вершинам, для которых flag=true, то общую асимптотику можно оценить как O(N+t), где t — количество

всех возможных вхождений всех строк-образцов в s. Если быть точным и учитывать вычисления автомата и суф. ссылок, то алгоритм работает O(M*k+N+t), где M=bohr.size(). Память — константные массивы размера k для каждой вершины бора, откуда и выливается оценка O(M*k).

Вывод

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..

ПРИЛОЖЕНИЕ А.

КОД ПРОГРАММЫ

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

#define SIZE 5

using namespace std;

struct numbers {
	long long int index;
	int pattern_num;
};

struct bohr_vertex {
	int next_vertex[SIZE];
	bool flag;
	int suff_link;
	int auto_move[SIZE];
	int par;
	char symbol;
	int suff_flink;
	int pattern_num[40];
};

vector<numbers> num;

class AhoCorasick{
private:
   vector <bohr_vertex> bohr;
   vector <string> pattern;
public:
bohr_vertex make_bohr_vertex(int par, char symbol) {
	bohr_vertex vertex;
	memset(vertex.next_vertex, 255, sizeof(vertex.next_vertex));
	vertex.flag = false;
	vertex.suff_link = -1;
	memset(vertex.auto_move, 255, sizeof(vertex.auto_move));
	vertex.par = par;
	vertex.symbol = symbol;
	vertex.suff_flink = -1;
	memset(vertex.pattern_num, 255, sizeof(vertex.pattern_num));
	return vertex;
}

void init_bohr() {
	bohr.push_back(make_bohr_vertex(-1, -1));
}
```

```
int find(char symbol) {
	int ch;
	switch (symbol)
	{
	case 'A':
		ch = 0;
		break;
	case 'C':
		ch = 1;
		break;
	case'G':
		ch = 2;
		break;
	case 'T':
		ch = 3;
		break;
	case 'N':
		ch = 4;
		break;
	default:
		break;
	}
	return ch;
}

void add_string_to_bohr(string s) {
	int num = 0;
	for (int i = 0; i < s.length(); i++) {
		char ch = find(s[i]);
		if (bohr[num].next_vertex[ch] == -1) {
			bohr.push_back(make_bohr_vertex(num, ch));
			bohr[num].next_vertex[ch] = bohr.size() - 1;
		}
		num = bohr[num].next_vertex[ch];
	}
	bohr[num].flag = true;
	pattern.push_back(s);
	for (int i = 0; i < 40; i++) {
		if (bohr[num].pattern_num[i] == -1) {
			bohr[num].pattern_num[i] = pattern.size() - 1;
			break;
		}
	}
}

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].symbol);  //пройдем по суф.ссылке предка и запустим переход по символу.
                }
        }
        return bohr[v].suff_link;
}

int get_auto_move(int v, char ch) {                     //вычисляемая функция
переходов
        if (bohr[v].auto_move[ch] == -1) {
                if (bohr[v].next_vertex[ch] != -1) {            //если из текущей
вершины еть ребро с символом ch
                        bohr[v].auto_move[ch] = bohr[v].next_vertex[ch];        //то идем по
нему
                }
                else {
                        if (v == 0) {
                                bohr[v].auto_move[ch] = 0;
                        }
                        else {
                                bohr[v].auto_move[ch] = get_auto_move(get_suff_link(v),
ch);  //иначе перейдем по суффиксальной ссылке
                        }
                }
        }
        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);  //если для
вершины по суф.ссылке flag=true, то это искомая вершина, иначе рекурсия.
                }
        }
        return bohr[v].suff_flink;
}

void check(int v, int i) {
        struct numbers s;
        for (int u = v; u != 0; u = get_suff_flink(u)) {
                if (bohr[u].flag) {
                        for (int j = 0; j < 40; j++) {
                                if (bohr[u].pattern_num[j] != -1) {
                                        s.index = i -
pattern[bohr[u].pattern_num[j]].length();
                                        s.pattern_num = bohr[u].pattern_num[j];
```

```
                                        num.push_back(s);
                                }
                                else
                                        break;
                        }
                }
        }
}

void find_all_pos(string s) {
        int u = 0;
        for (int i = 0; i < s.length(); i++) {
                u = get_auto_move(u, find(s[i]));
                check(u, i + 1);
        }
}
};
int main() {
   AhoCorasick ac;
        vector<string> patterns;  //подстроки при делении по джокеру
        vector<int> patterns_pos; //позиции подстрок
        string text;
        string temp;
        char joker;
        string pat;
        cin >> text >> temp >> joker;
        ac.init_bohr();
        for (int i = 0; i < temp.length(); i++) {
                if (temp[i] != joker) {
                        patterns_pos.push_back(i + 1);
                        for (int j = i; temp[j] != joker && j != temp.length(); j++) {
                                pat += temp[j];
                                i++;
                        }
                        ac.add_string_to_bohr(pat);
                        patterns.push_back(pat);
                        pat.clear();
                }
        }
        ac.find_all_pos(text);
        vector<int> c(text.length(), 0);
        for (int i = 0; i < num.size(); i++) {
                if (num[i].index < patterns_pos[num[i].pattern_num] - 1) continue;
                c[num[i].index - patterns_pos[num[i].pattern_num] + 1]++;
                if (c[num[i].index - patterns_pos[num[i].pattern_num] + 1] == patterns.size() &&
                        num[i].index - patterns_pos[num[i].pattern_num] + 1 <= text.length() -
temp.length())
                        cout << num[i].index - patterns_pos[num[i].pattern_num] + 2 << endl;
        }
        return 0;
}
```

ПРИЛОЖЕНИЕ Б.
ТЕСТОВЫЕ СЛУЧАИ

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования.

Ввод	Вывод
CCCA 1 CC	1 1 2 1
aaaa 3 a aa aaa	1 1 1 2 2 1 1 3 2 2 3 1 2 3 3 2 4 1
PoroshenkoZelenskyPutin 2 Poroshenko Putin	1 1 19 2