

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

Бергалиев М.

Преподаватель

Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Реализовать и исследовать алгоритм Ахо-Корасик поиска набора образцов в строке.

Постановка задачи

Реализовать алгоритм Ахо-Корасик и с его помощью для $n(1 \leq n \leq 3000)$ заданных шаблонов $P_i (|P_i| \leq 75)$ и текста $T (1 \leq |T| \leq 100000)$ найти все вхождения P_i в T .

Ход работы

Была написана программа на языке программирования C++. Код представлен в приложении А.

Сначала для набора шаблонов строится префиксное дерево(бор). Далее в дерево добавляются суффиксные ссылки проходом по суффиксным ссылкам родителя и поиска ребра, помеченного тем же символом, что и ребро из родителя в ребенка. Таким образом строится конечный автомат. Автомату передается текст для поиска всех шаблонов. Начиная с корня обходим дерево, переходя по ребрам, помеченным считанным из строки символом. При переходе проверяется, является ли данное состояние конечным и ищутся конечные вершины по суффиксным ссылкам. Для конечных вершин высчитывается положение начала вхождения шаблона в строку и записывается в ответ.

Тестирование

Тестирование проводилось в Ubuntu 16.04 LTS. По результатам тестирования были выявлены ошибки в коде. Тестовые случаи представлены в приложении Б.

Исследование алгоритма

В исследовании проводилась проверка теоретической оценки сложности алгоритма. По полученным данным был построен график,

оценивающий сложность алгоритма, показанный на рис. 1. По результатам сложность алгоритма $O(n \log |A| + |T| + k)$, где n - общая длина всех шаблонов, $|A|$ -- размер алфавита, $|T|$ - длина текста, в котором производится поиск, k - общая длина всех совпадений.

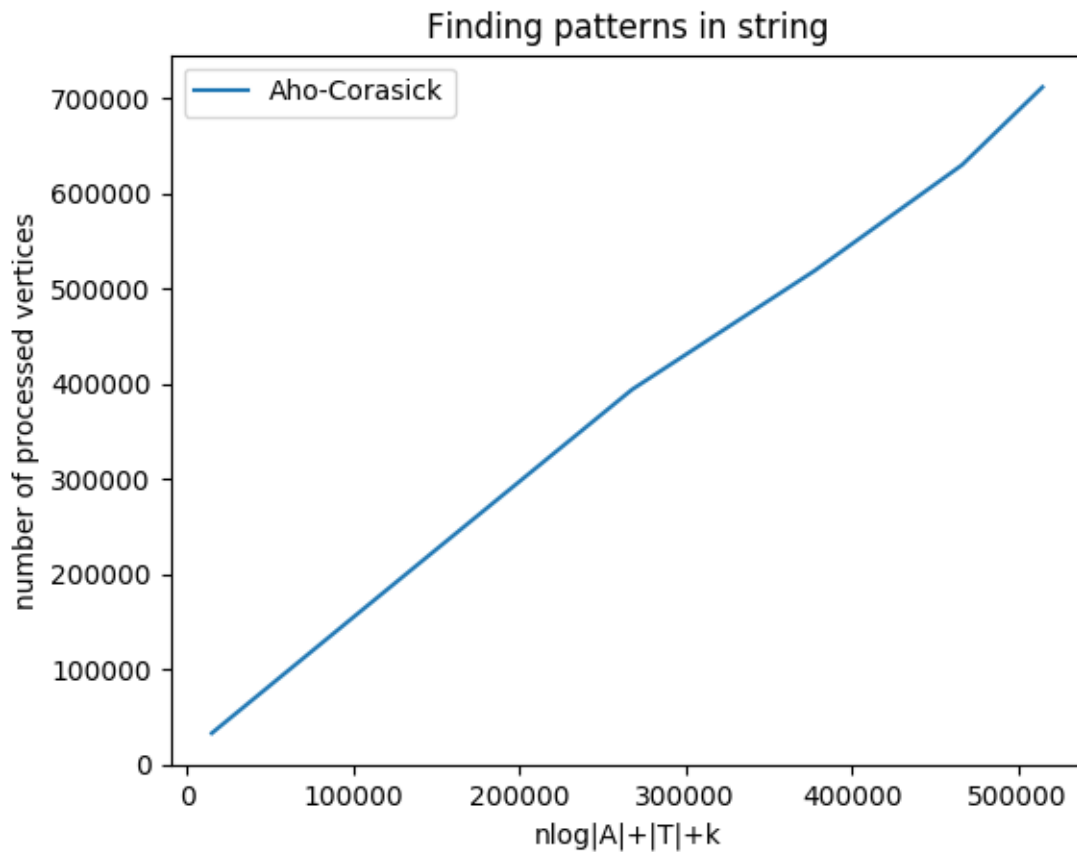


Рисунок 1 — Сложность алгоритма

Вывод

Был реализован и исследован алгоритм Ахо-Корасик. Сложность по количеству просмотренных вершин равна $O(n \log |A| + |T| + k)$.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <map>
#include <vector>

class trie{
    struct node{
        int n_pattern;
        int length;
        std::map<char, node*> direct;
        node* suff;
        node(int length) : suff(nullptr), length(length),
n_pattern(0) {}
    };
public:
    trie(const std::vector<std::string>& patterns){
        root = new node(0);
        node* cur;
        std::vector<std::map<std::string, node*>> nodes;
        int n;
        std::string cur_str;
        std::string s;
        for(int j=0; j<patterns.size(); ++j){
            s = patterns[j];
            cur = root;
            n = s.length();
            cur_str = "";
            for(int i=0; i<n; ++i){
                if(i >= nodes.size())

nodes.push_back(std::map<std::string, node*>());
                cur_str.push_back(s[i]);
                if(nodes[i].find(cur_str) ==
nodes[i].end()){
                    node* new_node = new node(i+1);
                    cur->direct.emplace(s[i],
new_node);
```

```

        cur = new_node;
        nodes[i].emplace(cur_str,
new_node);
    }
    else{
        cur->direct.emplace(s[i],    nodes[i]
[cur_str]);
        cur = nodes[i][cur_str];
    }
}
cur->n_pattern = j+1;
}
n = nodes.size();
node* suff;
for(auto k : nodes[0])
    k.second->suff = root;
for(int i=0; i<n; ++i)
    for(auto k : nodes[i]){
        for(auto j : k.second->direct){
            cur = k.second->suff;
            j.second->suff = root;
            while(cur != nullptr){
                if(cur->direct.find(j.first)    !=
cur->direct.end()){
                    j.second->suff    =    cur-
>direct[j.first];
                    break;
                }
            }
            cur = cur->suff;
        }
    }
}
std::vector<std::pair<int,
int>>
aho_corasick(std::string s){
    node* cur = root;
    node* cur_suff;
    std::vector<std::pair<int, int>> ans;
    for(int i=0; i<s.length(); ++i){

```

```

        if(cur->direct[s[i]] != nullptr){
            cur = cur->direct[s[i]];
            if(cur->n_pattern != 0)
                ans.push_back(std::make_pair(i-cur-
>length+2, cur->n_pattern));
            cur_suff = cur->suff;
            while(cur_suff != nullptr){
                if(cur_suff->n_pattern != 0)

                ans.push_back(std::make_pair(i-cur_suff->length+2,
cur_suff->n_pattern));

                cur_suff = cur_suff->suff;
            }
        }
        else if(cur != root){
            cur = cur->suff;
            --i;
        }
    }
    return ans;
}

```

private:

node* root;

};

int main(){

std::vector<std::string> v;

std::string s;

int n;

std::cin >> n;

for(int i=0; i<n; ++i){

std::cin >> s;

v.push_back(s);

}

trie t(v);

std::string text;

std::cin >> text;

```
        auto ans = t.aho_corasick(text);
        for(auto i : ans)
            std::cout << i.first << ' ' << i.second <<
std::endl;
        return 0;
    }
```

ПРИЛОЖЕНИЕ Б

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

Входные данные	Результат
6	1 1
a	1 2
ab	1 4
bc	2 3
abc	3 5
c	4 1
caa	3 6
abcaab	5 1
	5 2
	1 1
	1 2
	2 1
	1 3
	2 2
	3 1
4	5 1
	5 2
a	6 1
aa	8 1
aaa	10 1
aaaa	10 2
aaabaababaaaa	11 1
	10 3
	11 2
	12 1
	10 4
	11 3
	12 2
	13 1
3	1 1
a	3 1
aa	3 2
baac	4 1
abaaacbaac	4 2
	5 1
	8 1
	8 2
	9 1

	73
--	----