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 _____ Медведев И. С.

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 контангенируя буквы из ребер в порядке обхода, мы получим строку, соответствующую этой вершине. В этой структуре мы используем поля int number – номер строки-образца, bool flag – бит, показывающий является ли вершина исходной строкой. std::map<char,int> edge – вершины, в которые мы можем пойти из данной, std::map<char,int> auto_move – запоминает переходы автомата. Переход выполняется по двум параметрам — текущей вершине v и символу ch. по которому нам надо сдвинуться из этой вершины. 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, int suffix_link – суффиксная ссылка, int parent – индекс вершины родителя, char symbol – символ на ребре от родителя к этой вершине. В классе AK хранятся std::vector <Vertex> bohr – вектор, для хранения вершин, std::vector <std::string> patterns – вектор для хранения строк-шаблонов, int counter_of_ver – счетчик вершин. Также реализованы некоторые методы: void add_string(const std::string& pattern) – метод, создающий бор и добавляющий строки в patterns, void find_all_pos(const std::string& str) – метод, который выполняет автоматные переходы и выводит ответ на экран, int get_suffix_link(int node) – возвращает суффиксальную ссылку для данной

вершины, int get_auto_move(int node, char symbol) – метод, который выполняет автоматные переходы.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 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

Проведем оценку сложности алгоритма. Структура данных map из STL реализована красно-черным деревом, а время обращения к его элементам пропорционально логарифму числа элементов. Следовательно вычислительная сложность $O((H + n)\log k + c)$, где H — длина текста, в котором производится поиск, n — общая длина всех слов в словаре, k — размер алфавита, c — общая длина всех совпадений. Сложность по памяти $O(H + n)$, т.к. память выделяется для вершин шаблонов и для хранения текста.

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 <map>
#include <vector>

struct Vertex{
    int number;
    bool flag;
    std::map<char,int> edge;
    std::map<char,int> auto_move;
    int suffix_link;
    int parent;
    char symbol;

    Vertex(int m_parent,char m_symbol) {
        flag = false;
        number = 0;
        suffix_link = -1;
        parent = m_parent;
        symbol = m_symbol;
    }
};

class AK{
    std::vector <Vertex> bohr;
    std::vector <std::string> patterns;
    int counter_of_pat;
public:
    AK(){
        bohr.push_back(Vertex(0,0));
        counter_of_pat = 1;
    }


    void add_string(const std::string& pattern){
        size_t tmp = 0;
        patterns.push_back(pattern);
        for (auto sym : pattern) {
            if(bohr[tmp].edge.find(sym) == bohr[tmp].edge.end()){
                bohr.push_back(Vertex(tmp, sym));
                bohr[tmp].edge[sym] = counter_of_pat++;
            }
            tmp = bohr[tmp].edge[sym];
        }
        bohr[tmp].flag = true;
        bohr[tmp].number = patterns.size();
    }


    void find_all_pos(const std::string& str){
        int tmp = 0;
        for(int i = 0; i < str.size(); i++){
            tmp = get_auto_move(tmp, str[i]);
            for(int node = tmp; node != 0; node = get_suffix_link(node)){
                if (bohr[node].flag){
                    int pat = bohr[node].number;
                    int position = i - patterns[pat-1].size()+2;
                    std::cout<< position << " " << pat << std::endl;
```

```
                }
            }
        }
    }

    int get_suffix_link(int node){
            if(bohr[node].suffix_link == -1)
                if(!node || !bohr[node].parent)
                    bohr[node].suffix_link = 0;
                else
                    bohr[node].suffix_link =
get_auto_move(get_suffix_link(bohr[node].parent), bohr[node].symbol);
            return bohr[node].suffix_link;
        }


    int get_auto_move(int node, char symbol){
            if(bohr[node].auto_move.find(symbol) ==
bohr[node].auto_move.end())
                if(bohr[node].edge.find(symbol) != bohr[node].edge.end())
                    bohr[node].auto_move[symbol]=bohr[node].edge[symbol];
                else
                    bohr[node].auto_move[symbol]= (!node) ? 0 :
get_auto_move(get_suffix_link(node),symbol);
            return bohr[node].auto_move[symbol];
        }
};

int main()
{
    AK ak;
    std::string text, pattern;
    int num;
    std::cin >> text;
    std::cin >> num;
    for(int i = 0; i < num; i++) {
        std::cin >> pattern;
        ak.add_string(pattern);
    }
    ak.find_all_pos(text);
    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