Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

**Отчёт**

**по курсовой работе по теме  
«Алгоритм Ахо-Корасик»**

Дисциплина: «Объектно-ориентированное программирование»

Студент гр. 3331506/20102 Цыпин Н. С.

Преподаватель Ананьевский М. С.

Санкт-Петербург

2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc200039672)

[АЛГОРИТМИЧЕСКАЯ ЧАСТЬ 4](#_Toc200039673)

[Постановка задачи 4](#_Toc200039674)

[Области применения 4](#_Toc200039675)

[Теоретические сведения 4](#_Toc200039676)

[Описание алгоритма 5](#_Toc200039677)

[Пример работы 5](#_Toc200039678)

[Комментарии к программному коду 6](#_Toc200039679)

[ЗАКЛЮЧЕНИЕ 7](#_Toc200039680)

[СПИСОК ЛИТЕРАТУРЫ 8](#_Toc200039681)

[ПРИЛОЖЕНИЕ 9](#_Toc200039682)

# ВВЕДЕНИЕ

В современном мире обработка текстовой информации является одной из ключевых задач в компьютерных науках. С ростом объемов данных традиционные алгоритмы поиска подстрок, такие как наивный метод или алгоритм Кнута-Морриса-Пратта [2], становятся недостаточно эффективными при работе с множеством шаблонов.

Алгоритм Ахо-Корасик, разработанный в 1975 году Альфредом Ахо и Маргарет Корасик [1], решает эту проблему, позволяя находить все вхождения множества подстрок в тексте за линейное время относительно его длины. Этот алгоритм сочетает в себе преимущества префиксного дерева (бора) и конечного автомата с "fail-ссылками" [5], что делает его незаменимым в таких областях, как биоинформатика, сетевые технологии и системы компьютерной безопасности [3].

Актуальность данной работы обусловлена возрастающими требованиями к скорости и эффективности обработки текстовых данных в реальном времени. В рамках курсовой работы была выполнена реализация алгоритма Ахо-Корасик на языке C++.

# АЛГОРИТМИЧЕСКАЯ ЧАСТЬ

## Постановка задачи

Основная задача курсовой работы заключается в реализации алгоритма Ахо-Корасик для эффективного поиска множества подстрок в тексте.

Конкретные цели работы включают:

1. Изучение теоретических основ алгоритма.
2. Разработку объектно-ориентированной реализации на C++.
3. Проверку корректности работы алгоритма на различных тестовых данных.

Критерии успешности реализации:

1. Корректность поиска всех вхождений заданных шаблонов.
2. Читаемость и модульность кода.

## Области применения

**Антивирусное ПО:** используется для поиска сигнатур вредоносного кода в файлах [4].

**Биоинформатика:** применяется для анализа последовательностей ДНК и белков, поиска генетических маркеров в геномах [5].

**Сетевые технологии:** позволяет обнаруживать ключевые слова в интернет-трафике (например, для фильтрации контента) [3].

**Системы обработки текстов:** используется для проверки орфографии и грамматики, машинного перевода [1].

## Теоретические сведения

#### **Определение**

Алгоритм Ахо-Корасик основан на двух ключевых структурах данных.

1. Префиксное дерево:

* Каждый узел представляет символ.
* Путь от корня до узла соответствует строке.

2. Конечный автомат с fail-ссылками:

* Позволяет переходить к наиболее длинному суффиксу при несовпадении символа.
* Fail-ссылки вычисляются через обход в ширину (BFS).

Пример: Для шаблонов ["he", "she", "his", "hers"] бор будет содержать узлы для каждого символа, а fail-ссылки обеспечат переходы, например, от "shi" к "hi".

## Описание алгоритма

1. Построение бора: каждый шаблон добавляется в дерево, общие префиксы объединяются.
2. Расчёт fail-ссылок: для каждого узла определяется узел-потомок с самым длинным совпадающим суффиксом.
3. Поиск в тексте: текст обрабатывается посимвольно с переходами по fail-ссылкам при несовпадении.

## Пример работы

Для проверки корректности были использованы тестовые данные:

1. **Простой случай:**
   * Шаблоны: ["cat", "tok"].
   * Текст: "vcatenoktok".
   * Ожидаемый результат: позиции 1 и 8.
2. **Перекрывающиеся шаблоны:**
   * Шаблоны: ["abc", "bc"].
   * Текст: "abcbc".
   * Ожидаемый результат: позиции 0, 1 и 3.

Во всех тестах алгоритм показал 100% точность.

## Комментарии к программному коду

Реализованный код представлен в приложении. Он имеет структуру, соответствующую базовым принципам ООП. Также код содержит все необходимые комментарии. Описание структуры алгоритма и основных реализованных методов представлено далее.

Структура «Node» является базовым строительным блоком алгоритма. Хранит дочерние узлы, fail-ссылку, ссылка на родителя и символ, по которому пришли в этот узел, флаг, указывающий, что узел является концом какого-то шаблона, индексы шаблонов, которые заканчиваются в этом узле.

Класс «AhoCorasick» хранит в себе логику построения автомата и поиска. Содержит в себе бор (префиксное дерево) и список всех добавленных шаблонов.

Функция «add\_pattern» добавляет шаблон в бор, строя цепочку узлов.

Функция «build\_links» вычисляет fail-ссылки через BFS.

Функция «search» осуществляет поиск с помощью fail-переходов.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был реализован и проанализирован алгоритм Ахо-Корасик, предназначенный для эффективного поиска множества подстрок в тексте. Алгоритм продемонстрировал высокую производительность благодаря использованию префиксного дерева (бора) в сочетании с автоматом, дополненным fail-ссылками.

Алгоритм успешно прошел тестирование на различных наборах данных, включая случаи с перекрывающимися шаблонами и большими объемами текста. Реализация позволяет легко расширять функционал.

Таким образом, алгоритм Ахо-Корасик остается одним из наиболее востребованных инструментов для задач полнотекстового поиска, сочетая в себе высокую скорость, надежность и универсальность.

# СПИСОК ЛИТЕРАТУРЫ

1. Aho A. V., Corasick M. J. Efficient string matching: an aid to bibliographic search //Communications of the ACM. – 1975. – Т. 18. – №. 6. – С. 333-340.
2. Охотин А. Математические основы алгоритмов, осень 2024 г. Лекция 6. Реализация алгоритма Кнута–Морриса–Пратта на конечном автомате. Алгоритм Ахо–Корасик. Сжатие данных: коды Хаффмана, арифметическое кодирование.
3. Седжвик Р. Алгоритмы на C++. Фундаментальные алгоритмы и структуры данных //М.: Вильямс. – 2013. – Т. 1056
4. Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология //СПб.: Невский Диалект. – 2003. – С. 590-591.
5. Гринченко А. С., Самойлова Е. А. АЛГОРИТМ АХО-КОРАСИК //Международная научно-техническая конференция молодых ученых БГТУ им. ВГ Шухова, посвященная 170-летию со дня рождения ВГ Шухова. – 2023. – С. 114-121.

# ПРИЛОЖЕНИЕ

#include <iostream>

#include <vector>

#include <string>

#include <queue>

#include <map>

#include <set>

using namespace std;

class AhoCorasick {

private:

    struct Node {

        map<char, int> next;

        int link = -1;

        int parent;

        char parent\_char;

        bool is\_terminal = false;

        vector<int> pattern\_indices;

    };

    vector<Node> trie;

    vector<string> patterns;

public:

    AhoCorasick();

    void add\_pattern(const string& pattern);

    void build\_links();

    vector<pair<int, int>> search(const string& text);

};

AhoCorasick::AhoCorasick() {

    trie.emplace\_back();

    trie[0].link = 0;

    trie[0].parent = -1;

}

void AhoCorasick::add\_pattern(const string& pattern) {

    int node = 0;

    size\_t i = 0;

    while (i < pattern.size()) {

        char c = pattern[i];

        if (trie[node].next.count(c) == 0) {

            trie[node].next[c] = trie.size();

            trie.emplace\_back();

            trie.back().parent = node;

            trie.back().parent\_char = c;

        }

        node = trie[node].next[c];

        i++;

    }

    trie[node].is\_terminal = true;

    trie[node].pattern\_indices.push\_back(patterns.size());

    patterns.push\_back(pattern);

}

void AhoCorasick::build\_links() {

    queue<int> q;

    q.push(0);

    while (!q.empty()) {

        int node = q.front();

        q.pop();

        map<char, int>::iterator it = trie[node].next.begin();

        while (it != trie[node].next.end()) {

            q.push(it->second);

            it++;

        }

        if (node == 0 || trie[node].parent == 0) {

            trie[node].link = 0;

        } else {

            int parent\_link = trie[trie[node].parent].link;

            char c = trie[node].parent\_char;

            while (parent\_link != 0 && trie[parent\_link].next.count(c) == 0) {

                parent\_link = trie[parent\_link].link;

            }

            if (trie[parent\_link].next.count(c)) {

                trie[node].link = trie[parent\_link].next[c];

            } else {

                trie[node].link = 0;

            }

        }

    }

}

vector<pair<int, int>> AhoCorasick::search(const string& text) {

    vector<pair<int, int>> matches;

    int node = 0;

    int pos = 0;

    while (pos < text.size()) {

        char c = text[pos];

        while (node != 0 && trie[node].next.count(c) == 0) {

            node = trie[node].link;

        }

        if (trie[node].next.count(c)) {

            node = trie[node].next[c];

        }

        int current = node;

        while (current != 0) {

            if (trie[current].is\_terminal) {

                size\_t j = 0;

                while (j < trie[current].pattern\_indices.size()) {

                    int pattern\_idx = trie[current].pattern\_indices[j];

                    matches.emplace\_back(pattern\_idx, pos - patterns[pattern\_idx].size() + 1);

                    j++;

                }

            }

            current = trie[current].link;

        }

        pos++;

    }

    return matches;

}

int main() {

    AhoCorasick ac;

    vector<string> patterns = {"he", "she", "his", "hers"};

    size\_t i = 0;

    while (i < patterns.size()) {

        ac.add\_pattern(patterns[i]);

        i++;

    }

    ac.build\_links();

    string text = "ushers";

    auto matches = ac.search(text);

    cout << "Текст: " << text << endl;

    cout << "Найденные шаблоны:" << endl;

    i = 0;

    while (i < matches.size()) {

        cout << "  " << patterns[matches[i].first] << " на позиции " << matches[i].second << endl;

        i++;

    }

    return 0;

}