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

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, становятся недостаточно эффективными при работе с множеством шаблонов.

Алгоритм Ахо-Корасик, разработанный в 1975 году Альфредом Ахо и Маргарет Корасик, 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", что делает его незаменимым в таких областях, как биоинформатика, сетевые технологии и системы компьютерной безопасности.

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.

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

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

Системы обработки текстов: используется для проверки орфографии и грамматики, машинного перевода.

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

```cpp
#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;
}
```