**Отчет по лабораторной работе №1**

по дисциплине «Построение и анализ алгоритмов»

**Тема: Реализация автомата Ахо-Корасик с использованием суффиксных ссылок**

**Студент группы 3388**

Раутио И.А.

**Преподаватель**

Жангиров Т.Р.

Санкт-Петербург  
2025

**Задание**

Реализовать алгоритм Ахо-Корасик для множественного поиска подстрок в строке. Алгоритм должен:

* Поддерживать добавление слов в словарь;
* Строить бор (trie);
* Находить все вхождения слов из словаря в заданной строке;
* Выводить позиции вхождений;
* Удалять найденные подстроки из строки;
* Определять максимальное количество исходящих дуг из узла бора.

Входные данные:

* Строка **T** — текст, в котором производится поиск.
* Целое число **n** — количество слов в словаре.
* **n** строк — слова, которые нужно найти в тексте.

Выходные данные:

* Все вхождения слов из словаря в тексте в формате: **позиция\_вхождения номер\_слова**;
* Максимальное количество исходящих дуг в любом узле бора;
* Строка **T**, из которой удалены найденные подстроки.

**Выполнение работы**

Для реализации задачи был использован алгоритм Ахо-Корасик , основанный на построении бора с использованием суффиксных ссылок и сжатых переходов . Это позволяет эффективно находить несколько шаблонов в строке за один проход.

**Работа алгоритма:**

1. Добавление слов в бор :
   * Для каждого слова строится путь в дереве.
   * Каждый символ слова добавляется как отдельный узел.
   * Последний узел помечается как терминальный и хранит само слово.
2. Построение суффиксных ссылок (**get\_link()**) :
   * Суффиксная ссылка для узла указывает на самый длинный собственный суффикс, соответствующий одному из слов из словаря.
   * Ссылки вычисляются с помощью BFS (обход уровня за уровнем).
3. Поиск всех вхождений (**find()**) :
   * Обрабатывается текст посимвольно.
   * Для каждого символа осуществляется переход по бору.
   * Если достигнут терминальный узел или найден через суффиксную ссылку — фиксируется вхождение соответствующего слова.
4. Удаление найденных подстрок (**remove\_patterns()**) :
   * После поиска вычисляются индексы символов, входящих в найденные подстроки.
   * Эти символы исключаются из исходной строки.
5. Определение максимального числа исходящих дуг (**max\_outgoing\_edges()**) :
   * Обход бора в ширину.
   * Для каждого узла считается количество дочерних узлов.
   * Фиксируется максимальное значение.

class Node {

public:

char val; // Символ, хранящийся в узле

unordered\_map<char, Node\*> next; // Дочерние узлы

Node\* parent; // Указатель на родителя

bool is\_terminal; // Является ли узел концом слова

string word; // Хранимое слово (если узел терминальный)

Node\* link; // Суффиксная ссылка

Node\* word\_link; // Ссылка на слово (для оптимизации вывода)

unordered\_map<char, Node\*> moves; // Переходы с учетом суффиксных ссылок

};

class Vertex {

public:

Node\* \_root; // Корень бора

unordered\_map<string, int> \_words; // Нумерация слов для вывода

int \_len; // Размер словаря

};

**Оптимизации алгоритма**

1. Кэширование суффиксных ссылок и переходов :
   * Чтобы не пересчитывать их при каждом запросе, результаты сохраняются в полях **link**, **word\_link**, **moves**.
2. Использование BFS для построения суффиксных ссылок :
   * Позволяет корректно обработать связи между уровнями дерева.
3. Эффективный поиск вхождений через сжатые переходы :
   * Используются суффиксные ссылки и word-ссылки для быстрого доступа к словам.

**Оценка сложности**

**Временная сложность:**

* Построение бора : O(M), где M — суммарная длина всех слов.
* Построение суффиксных ссылок : O(M).
* Поиск в тексте : O(N + K), где N — длина текста, K — количество вхождений.
* Удаление подстрок : O(N).

Итоговая временная сложность: O(M + N + K) .

**Пространственная сложность:**

* Бор : O(M) узлов.
* Суффиксные ссылки и переходы : O(M).
* Массив вхождений : O(K).

Итоговая пространственная сложность: O(M + K) .

**Пример выполнения**

**Входные данные:**

abracadabra

5

a ab cad cbrabrabra

**Выходные данные:**

1 1

1 2

5 3

7 4

11 1

3

Brdabra

**Разбор:**

* Найдены вхождения:
  + **'a'** на позиции 1,
  + **'ab'** на позиции 1,
  + **'cad'** на позиции 5,
  + **'cbrabrabra'** на позиции 7,
  + **'a'** на позиции 11.
* Максимальное число исходящих дуг: 3.
* После удаления всех найденных подстрок осталось: **"brdabra"**.

**Выводы**

В ходе лабораторной работы был исследован и реализован алгоритм Ахо-Корасик , позволяющий эффективно искать множество подстрок в тексте за один проход.

Алгоритм показал высокую эффективность благодаря использованию:

* Бора для представления словаря;
* Суффиксных ссылок для быстрого перехода между состояниями;
* Кэширования для ускорения поиска.

Программа корректно обрабатывает входные данные и выполняет все требуемые операции: поиск, нумерацию, удаление и анализ структуры бора.