МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Построение и Анализ Алгоритмов»
Тема: «Алгоритм Ахо-Корасик»

Студент гр. 7304 _____ Петруненко Д.А

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2019

Цель работы

Освоить алгоритм Ахо – Карасик, который реализует поиск множества подстрок из словаря в данной строке.

Задания лабораторной работы

1. Разработать программу, решающую задачу точного поиска набора образцов.
2. Используя реализацию точного множественного поиска, решить задачу точного поиска для одного образца с джокером

Основные теоретические положения

Бор (trie) — структура данных для хранения набора строк, представляющая из себя подвешенное дерево с символами на рёбрах. Строки получаются последовательной записью всех символов, хранящихся на рёбрах между корнем бора и терминальной вершиной. Размер бора линейно зависит от суммы длин всех строк, а поиск в бору занимает время, пропорциональное длине образца.

Детерминированный конечный автомат — набор из пяти элементов $\langle \Sigma, Q, s \in Q, T \subset Q, \delta : Q \times \Sigma \to Q \rangle \langle \Sigma, Q, s \in Q, T \subset Q, \delta : Q \times \Sigma \to Q \rangle$, где Σ — алфавит (англ. alphabet), Q — множество состояний (англ. finite set of states), s — начальное (стартовое) состояние (англ. start state), T — множество допускающих состояний (англ. set of accept states), δ — функция переходов (англ. transition function).

Пусть $x\alpha$ обозначает произвольную строку, где x — её первый символ, а α — оставшаяся подстрока (возможно пустая). Если для внутренней вершины v с путевой меткой $x\alpha$ существует другая вершина $s(v)$ с путевой меткой α, то ссылка из v в $s(v)$ называется суффиксной ссылкой (англ. suffix link).

Ход работы

1. Алгоритм Ахо-Корасик

1.1. На основе набора паттернов строим префиксное дерево.

1.2. Рассматриваем префиксное дерево как конечный детерминированный автомат. Стартовая позиция в корне.

1.3. Считываем первый символ текста.

1.4. Переходим в следующее состояние по ребру, обозначающему этот символ. Если такого ребра нет, то идем по суффиксной ссылке. Если суффиксной ссылки нет, то запоминаем символ, по которому пришли в данный узел, берем суффиксную ссылку родителя и пытаемся перейти по этому символу. Данный шаг выполняется рекурсивно, пока не найден такой переход или не достигнут корень.

1.5. Выполняем шаг 4 до тех пор, пока не найдем валидный переход по текущему символу текста или пока не достигнем корня.

1.6. Проверяем является ли текущее состояние каким-либо паттерном. Для этого переходим по суффиксным ссылкам, проверяя соответствующий флаг. Если это паттерн, выводим его номер и позицию в тексте в консоль.

1.7. Если не конец текста переходим к шагу 3.

2. Алгоритм Ахо-Корасик + паттерн с джокером

2.1. Разбиваем паттерн на части, разделенные джокерами. Запоминаем их позицию в паттерне (индекс).

2.2. Используя алгоритм Ахо-Корасик ищем позицию этих паттернов в тексте.

2.3. Создаем массив нулей, размер которого совпадает с длиной текста.

2.4. Для каждого вхождения паттерна инкрементируем $i - j + 1$ позицию в массиве, где i – индекс вхождения, j – позиция данного паттерна в исходном паттерне.

2.5. Таким образом, в тех позициях массива, где значение совпадает с количеством паттернов, присутствует совпадение с исходным паттерном.

Результат работы алгоритма Ахо-Корасик

1. Входные данные:
ATATAT
1
AT
Выходные данные:
1 1
3 1
5 1
2. Входные данные:
ATTATNA
2
AT
TN
Выходные данные:
1 1
1 4
2 5

Результат работы алгоритма Ахо-Корасик с поиска шаблонов с масками

1. Входные данные:

ACAGA

A?A

?

Выходные данные:

1

3

2. Входные данные:

ATATNATNT

T

$

Выходные данные:

1
3
6

Вывод

В ходе данной лабораторной работы был реализован алгоритм Ахо-Карасика на языке c++. Данный алгоритм делает точный поиск набора образцов в строке. В нем используются такие понятия, как бор, конечный детерминированный автомат, суффиксальные ссылки. Построение бора происходит за время O(n*logK), так как map<char,int> next, а не массив. Поиск происходит за время O(n*Len),где Len – это длина строки. Можно сделать оптимизацию, введя понятие хороших суффиксальных ссылок, тогда поиск будет занимать время O(Len).