Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Отчёт

по курсовой работе

Дисциплина: Объектно-ориентированное программирование

Тема: Aho-Corasick algorithm

Студент гр. 3331506/90401 Вестников Р. Н.

Преподаватель Ананьевский М. С.

Санкт-Петербург
2022

ВВЕДЕНИЕ

Алгоритм Ахо – Корасик — алгоритм поиска подстроки, разработанный Альфредом Ахо и Маргарет Корасик в 1975 году, реализует поиск множества подстрок из словаря в данной строке. Широко применяется в системном программном обеспечении, например, используется в утилите поиска *grep*.

Задача алгоритма: Дан набор строк в алфавите размера k суммарной длины m. Необходимо найти для каждой строки все её вхождения в текст.

Описание алгоритма

Пусть дан набор строк $s_1, s_1, \dots, s_m$ алфавита размера k суммарной длины n, называемый словарем, и длинный текст t. Необходимо определить, есть ли в тексте хотя бы одно слово из словаря, и если есть, то на какой позиции.

Для примера будем искать в строке "*aabccbbbabcdaddcd*" следующие подстроки:

1) "*abc*"
2) "*bbb*"
3) "*add*"
4) "*cd*"
5) "*c*"
6) "*abcd*"

Реализация алгоритма осуществляется по шагам.

Первый шаг: построим по словарю бор (префиксное дерево).

Бор – 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.

Бор для рассматриваемого словаря представлен на рис. 1.

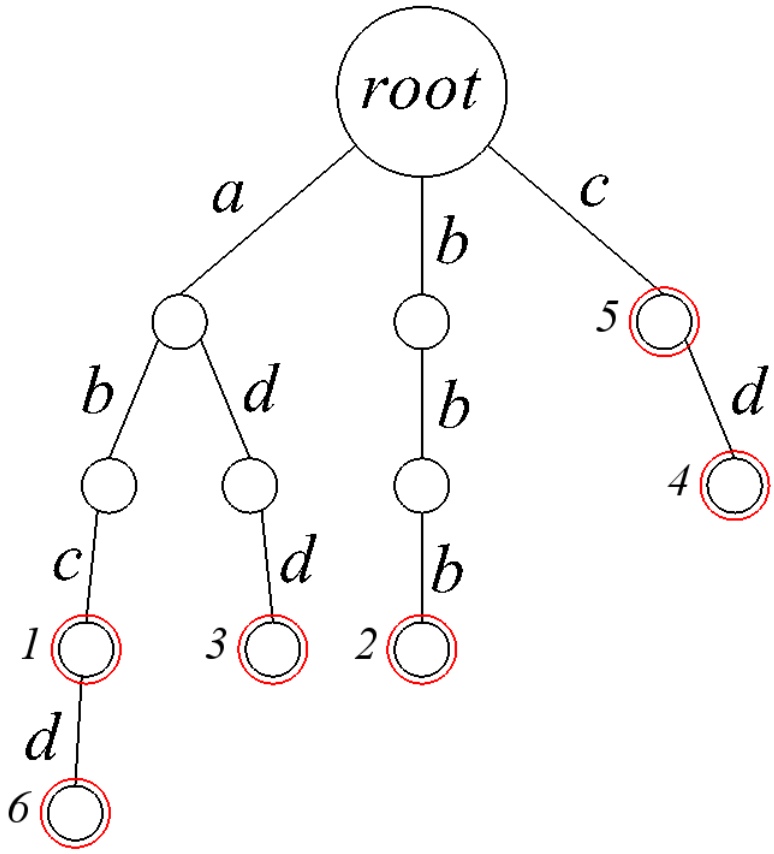

Рис. 1. Пример префиксного дерева (бора)

Второй шаг: построим конечный детерминированный автомат.

Конечный автомат – математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных. Является частным случаем абстрактного дискретного автомата, число возможных внутренних состояний которого конечно.

Если мы рассмотрим любую вершину бора, то строка, которая соответствует ей, является префиксом одной или нескольких строк из набора. Т.е. каждую вершину бора можно понимать как позицию в одной или нескольких строках из набора.

Фактически, вершины бора можно понимать как состояния конечного детерминированного автомата. Находясь в каком-либо состоянии, мы под воздействием какой-то входной буквы переходим в другое состояние — т.е. в

другую позицию в наборе строк.

Т.е. мы можем понимать рёбра бора как переходы в автомате по соответствующей букве. Однако одними только рёбрами бора нельзя ограничиваться. Если мы пытаемся выполнить переход по какой-либо букве, а соответствующего ребра в боре нет, то мы тем не менее должны перейти в какое-то состояние. Для этого нам нужны *суффиксные ссылки*.

Суффиксная ссылка для каждой вершины p – это вершина, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине p. Единственный особый случай – корень бора, для удобства суффиксную ссылку из него проведём в себя же. Теперь мы можем переформулировать утверждение по поводу переходов в автомате так: пока из текущей вершины бора нет перехода по соответствующей букве (или пока мы не придём в корень бора), мы должны переходить по суффиксной ссылке.

Таким образом, мы свели задачу построения автомата к задаче нахождения суффиксных ссылок для всех вершин бора. Однако строить эти суффиксные ссылки мы будем, как ни странно, наоборот, с помощью построенных в автомате переходов.

Заметим, что если мы хотим узнать суффиксную ссылку для некоторой вершины v, то мы можем перейти в предка par текущей вершины (пусть $symb$ – буква, по которой из par есть переход в v), затем перейти по его суффиксной ссылке, а затем из неё выполнить переход в автомате по букве $symb$. Иллюстрация данного действия представлена на рис. 2.

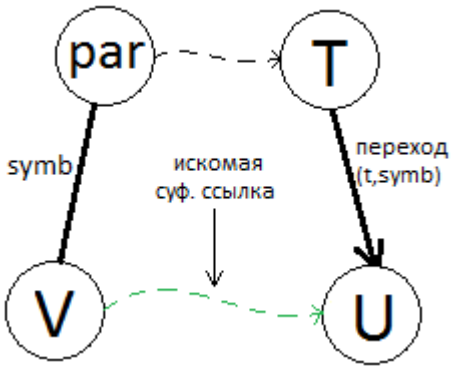

Рис. 2. Нахождение суффиксной ссылки

Таким образом, задача нахождения перехода свелась к задаче нахождения суффиксной ссылки, а задача нахождения суффиксной ссылки – к задаче нахождения суффиксной ссылки и перехода, но уже для более близких к корню вершин. Мы получили рекурсивную зависимость, но не бесконечную, и, более того, разрешить которую можно за линейное время.

Реализация бора с расставленными суффиксными ссылками представлена на рис. 3.

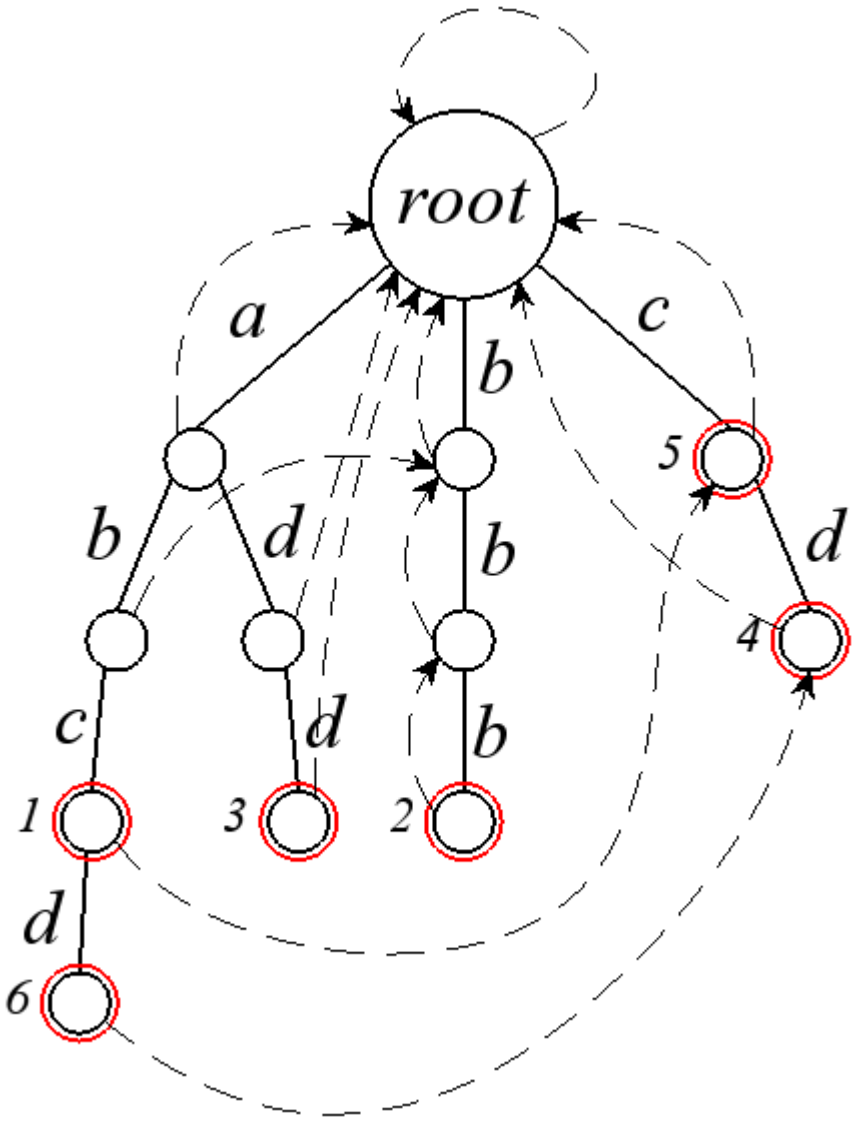

Рис. 3. Бор с расставленными суффиксными ссылками

Третий шаг: Построение сжатых суффиксных ссылок.

С автоматом несложно определить сам алгоритм: считываем строку, двигаемся из состояния в состояние по символам строки, в каждом из состояний двигаемся по суффиксным ссылкам, то есть по суффиксам строки в позиции автомата, проверяя при этом наличие их в боре.

Все бы ничего, но оказывается, что этот вариант Ахо-Корасика имеет квадратичную асимптотику относительно длины считываемой строки. Наша цель в том, чтобы двигаясь по суффиксным ссылкам попадать только в заведомо имеющиеся среди строк-образцов. Для этого введем понятие сжатых суффиксных ссылок. Сжатая суффиксная ссылка – это ближайший суффикс, имеющийся в боре, для которого $flag == true$. Число «скачков» при использование таких ссылок уменьшится и станет пропорционально количеству искомых вхождений, оканчивающихся в этой позиции. Аналогично обычным суффиксным ссылкам сжатые суффиксные ссылки могут быть найдены при помощи ленивой рекурсии.

Визуализация сжатых ссылок в сравнении с обычными суффиксными ссылками представлена на рис.4. Как видно из рисунка, переход по сжатой ссылке значительно быстрее перехода по нескольким суффиксным.

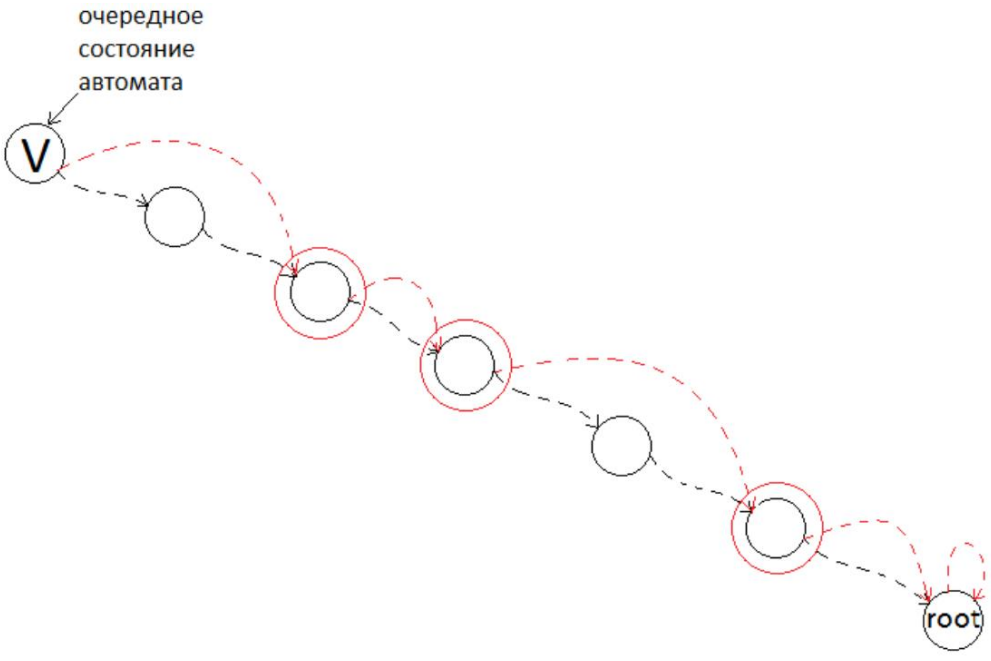

Рис. 4. Сжатые суффиксные ссылки

Результат применения алгоритма для исследуемой строки:

```
2 abc
4 c
5 c
6 bbb
9 abc
11 c
9 abcd
11 cd
13 add
16 c
16 cd
```

Как видим алгоритм успешно нашёл все возможные подстроки в исходной строке и указал их место.

Исследование алгоритма

Произведём замеры времени выполнения алгоритма от размера входных данных (размера строки). Будем постепенно увеличивать входные данные при неизменном алфавите ($k = 10$) и искомых строках (бор не меняется). Зависимость скорости алгоритма от размера строки представлена в таблице 1 и на рис. 5.

Таблица 1 – Исследование скорости алгоритма

Число символов исследуемой строки, млн	Время выполнения алгоритма, с
1	0,062
2	0,156
3	0,171
4	0,218
5	0,281
6	0,322
7	0,375
8	0,406
9	0,467
10	0,498

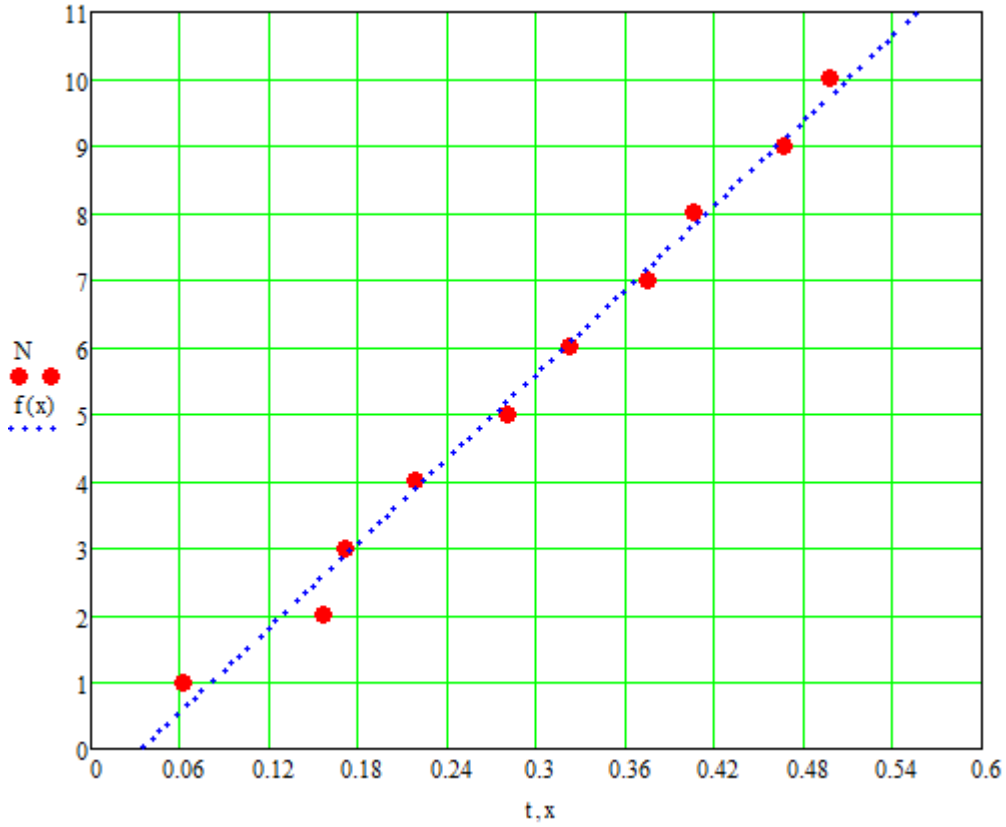

Рис. 5. График результатов исследования скорости

Аппроксимировав все точки, можем видеть линейную зависимость времени от размера входных данных. Стоит повторить, что данная зависимости справедлива исключительно при одном и том же значении алфавита и искомых строках.

Заключение

Список литературы

ПРИЛОЖЕНИЕ