Лабораторная работа № 5 по курсу дискретного анализа: Суффиксные деревья

Выполнил студент группы 8О-207Б-17 МАИ *Ваньков Денис.*

Условие

Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Построив такое дерево для некоторых из входных строк, необходимо воспользоваться полученным суффиксным деревом для решения своего варианта задания. Алфавит строк: строчные буквы латинского алфавита (т.е., от a до z).

Упрощенный вариант:

Реализовать поиск подстрок в тексте с использование суффиксного дерева. Суффиксное дерево можно построить за $O(n^2)$ наивным методом. Задание, алфавит, входные/выходные данные и пример такие же, как в первом варианте.

Вариант задания: 4. Линеаризация циклической строки

Линеаризовать циклическую строку, то есть найти минимальный в лексикографическом смысле разрез циклической строки.

Входные данные: некий разрез циклической строки.

Выходные данные: минимальный в лексикографическом смысле разрез.

Метод решения

Реализовать алгоритм Укконена, строящий суффиксное дерево за линейное время. Для этого использовать 2 оптимизации.

Оптимизация №1: был листом, листом и останешься. При добавлении символа к листу получаем новый лист. Поэтому пусть, мы создаём лист не только для рассмотренной части строки, а для всей все строки до конца. Для этого правой границе подстроки, соответствующей ребру до листа, сопоставим бесконечность. Таким образом, пропадает необходимость каждый раз продлевать листы. В итоге, если мы для какого-то суффикса создали лист, то этот суффикс мы больше рассматривать не будем. Понятно, что на асимтотику это не влияет, но избавляет нас от ненужных действий.

Оптимизация №2: если просто прошли по ребру, то и в меньших суффиксах мы тоже пройдём по ребру. У нас есть 3 действия с вершинами при добавлении символа: 1) Продление листа (выполняется для листа) 2) Создание развилки (выполняется для мнимой вершины или развилки, из которых нет ребра с этим символом) 3) Просто проход по ребру (выполняется для мнимой вершины или развилки, из которых есть такое ребро) Утверждается, что если рассматривать суффиксы по уменьшеню длины, то действия с ними будут выполняться в таком порядке 11...12...23..33. (Это следует из того, что из вершины суффикса данной вершины есть хотя бы все рёбра, аналогичные

рёбрам из данной вершины, но ещё могут быть другие рёбра). Причём позиции смены действия 1 на 2 и 2 на 3 двигаются только влево (не забываем, что после каждой фазы у нас длина этой последовательности увеличивается на 1, т.к. добавляется новый суффикс).

Описание программы

Первоначально я создал класс SuffixTreeNode, в котором описал конструктор, построение суффиксного дерева с помощью алгоритма Укконена, а также функцию GetLinear CircleString, которая и является ответом на поставленную задачу. Также у меня имеется массив потомков, от размера алфавита children[ALPHABET_SIZE] типа shared pointer, и suffix_link тогоже типа. Затем создал структуру ActivePoint, которая состоит из вершины типа shared pointer, символа и длины формата int.

Нужно описать функцию продления дерева ExtendSuffixTree, котрая будет оптимально за линию увеличивать длину нашего дерерва. В этой функции содержится основная часть алгоритма Укконена.

И, теперь нужно описать функцию ответа - GetLinearCircleString. Чтобы выполнить поставленную задачу оптимально и за линейное время мне нужно входную строку p длины n увеличить вдвое таким образом, что за исходной идет она же, без разделителей. Таким образом, я имеею строку pp, длины $2n$, и для этой строки строю суффиксное дерево за $O(2n)$. Затем, осталось только произвести лексикографический поиск по этому дерево, чтобы найти наименьшую строку, которая и будет ответом.

Дневник отладки

В процессе отладки долго не получалось понять почему дерево строиться не совсем корректно, однако все же удалось отловить эту ошибку и исправить.

Выводы

Для меня эта лабораторная работа оказалась одной из самых сложных, как в плане понимания, так и в плане реализации. Не достаточно было только понять как работает этот алгоритм, но нужно было еще и придумать как это написать. Однако все же удалось с помощью различных источников реализовать поставленную задачу. Дерево корректно строится за $O(n)$, доказательство этого лежит в доказательстве корректности алгоритма Укконена.

Т.к. переход по прямой ссылке работает за O(1), а их не более N, то достаточно только рассмотреть переход вниз по рёбрам. Для этого рассмотрим длину смещения по ребру, по которому мы поднялись к предку. Пусть она равна L. Тогда при каждом прохождении во рёбрам вниз эта длина уменьшается хотя бы на 1. А увеличивается она только в основоной процедуре при проходе по ребру по символу C, причём не более чем на 1 за раз. А таких проходов N. Конечное значение смещения лежит в диапазоне $[0, N]$, количество удлинений N, то и суммарное количество сокращений может быть не

более $2N$. Значит, суммарное количество продвижений вниз при переходе по суффикной ссылке $O(3N)$. В итоге, количество каждого вида действий $O(N)$, а значит и суммарное количество всех действий $O(N)$.