Лабораторная работа № 5 по курсу Дискретный Анализ. Суффиксные деревья

Выполнил студент группы М8О-307Б-21 МАИ *Друхольский Александр.*

Условие

Задача:

> Реализовать поиск подстрок в тексте с использование суффиксного дерева. Суффиксное дерево можно построить за $O(n^2)$ наивным методом.

Метод решения

> Для решения задчи реализуем наивный метод потсроения суффиксного дерева. Однако на ребрах мы будем хранить строки в неявном виде: *start* и *end* - позиции начала и конца подстроки в строке. Информация о рёбрах будет храниться в вершинах, так как мы можем провести однозначное соответствие между вершиной и ребром, исключая корень дерева. Нам требуется построить compact trie. Мы используем map для хранения указателей на детей, что позволит нам быстрее производить поиск по дереву.
>
> Строить дерево мы будем начиная с самого длинного суффикса исходной строки и до суффикса длиной 1. Для построения мы просто идём по дереву, пока не найдём позицию, с которой нам нужно вставить часть нового суффикса. Использование позиций вместо явного хранения строк и позволит построить дерево за $O(n^2)$
>
> Для поиска по сути реализован похожий на построение алгоритм и алгоритм поиска в глубину. При проходе по дереву если подстрока найдена (ищем быстро благодаря map), то мы запускаем поиск в глубину, чтобы посчитать индекс вхождений подстроки в строку. Если мы в целом нашли подстроку в каком либо из ребер, то все вхождения этой строки будут принадлежать дочерним листовым ребрам, то есть все вхождения подстроки принадлежат суффиксам строки. В худшем случае сложность поиска $O(m * log(n))$, где n - длина строки, m - длина паттерна, для которого ищем вхождение.

Описание программы

Основные моменты:

1. Для представления вершины с ребром реализован *class Node*, в котором определены основные методы Find - поиск для вставки, CreateNode - создание новой вершины, Split - деление ребра (по сути мы не делим ребро, а переопределяем его, заменяя его детей на двух новых и передавя одному из детей прошлых детей этой вершины), Search и Dfs - для поиска вхождений.

2. Взаимодействие с деревом происходит с помощью *class SFTree*. В нём реализованы методы, которые вызывают методы для Node. Это сделано, чтобы при использовании взаимодействовать только с объектом дерева, а не самой вершины, что более логично.

Дневник отладки

CE - 1 Выбрал не тот компилятор.

local WA Ошибки при подсчёте индекса в алгоритме поиска, что было вызвано рассмотрением не всех случаев.

Тест производительности

Исследуем построение дерева на сложность $O(n^2)$

№	Длина строки	Время, с
1	10	0.000033
2	100	0.000317
3	1000	0.002795
4	10000	0.023227
5	100000	0.267882

В данном примере квадратичная сложность явно не наблюдается. На самом деле она проявится в худшем случае для алгоритма, когда строка не рандомная, а состоит из повторяющихся символов. Проверим алгоритм на строках вида аааааа.

№	Длина строки	Время, с
1	10	0.000073
2	100	0.00077
3	1000	0.011288
4	10000	0.508539
5	100000	32.321

В данном примере время работы алгоритма уже больше похоже на квадратичное. При увелчении длины в 10 раз, время работы возрастает в среднем на два порядка.

Выводы

Я разработал алгоритм построения суффиксного дерева наивным методом за $O(n^2)$, используя улучшение с хранением индексов вместо строк. Я узнал для чего нужны такие деревья и в чём удобство их использования.