Лабораторная работа № 5 по курсу дискретного анализа: Суффиксные деревья

Выполнила студентка группы 08-308 МАИ *Шевлякова София.*

Условие

Реализовать поиск подстрок в тексте с использование суффиксного дерева. Суффиксное дерево можно построить за O(n^2) наивным методом.
Формат ввода: текст располагается на первой строке, затем, до конца файла, следуют строки с образцами.
Формат вывода: для каждого образца, найденного в тексте, нужно распечатать строчку, начинающуюся с последовательного номера этого образца и двоеточия, за которым, через запятую, нужно перечислить номера позиций, где встречается образец в порядке возрастания.

Метод решения

Наивный способ представляет собой compact trie, который построен по всем суффиксам текущей строки, сначала помещается несобственный суффикс S[1..m]$, затем последовательно вводятся суффиксы S[i..m]$ для i от 2 до m.
При этом мы должны сохранить свойства суффиксного дерева:

- Количество листьев в trie совпадает с количеством суффиксов в строке
- Trie должен оставаться compact trie

В наивном алгоритме при вставке очередного суффикса сравниваем первую букву суффикса и детей корня, если совпадений нет, то есть из корня мы не можем перейти ни по одному из ребер, то вставляем весь суффикс от корня. Иначе, идем по совпавшему ребру до того момента, пока встретим несовпадение. Тогда нам будет необходимо сделать split текущего ребра, то есть уменьшить его правую границу, создать 2 новых ребра, одно из которых будет отвечать за продолжение исходной дуги, ему необходимо будет перекопировать детей, а второе будет являться оставшейся частью суффикса. Если мы дошли до конца ребра и не встретили несовпадений, то ищем среди детей этого ребра букву, по которой можем перейти дальше, если общая буква существует, то повторяем алгоритм заново, если нет, то вставляем оставшийся суффикс от текущего ребра.
Поиск паттерна работает аналогично вставке, только когда мы встречаем несовпадение, то просто останавливаем цикл, если же мы не встретили несовпадений и при этом паттерн закончился, то мы нашли вхождение. Чтобы узнать на каких позициях есть вхождение, необходимо от текущей вершины обойти всех детей поиском в глубину. Так мы получим массив, в котором будут хранится индексы вхождений паттерна в текст.

Описание программы

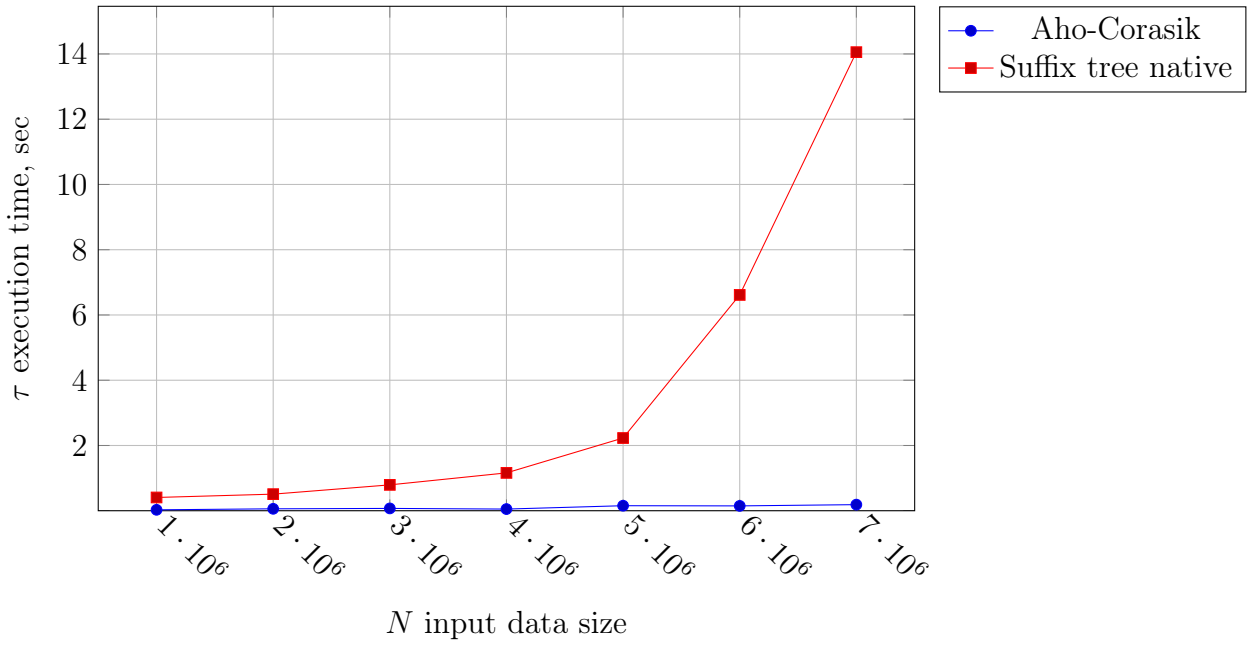
В данной программе содержится один файл main.cpp, в котором реализован наивный алгоритм построения суффиксного дерева.
Основные этапы работы программы:

- ReadText() — функция считывает текст, добавляет $ в конец строки и для каждого суффикса текста вызывает функцию Insert()
- Insert() — функция выполняет вставку суффикса в trie
- NewVertex() — функция создает новое ребро, как элемент массива. Существует вариант этой функции с заполнением поля структуры enter, если мы знаем, что это ребро будет являться листом
- ReadPattern() — функция считывает паттерн, вызывает для него функцию Search() и выводит результат
- Search() — функция выполняет поиск паттерна в тексте, если паттерн является подстрокой текста, то вызывается функция DFS() для обхода в глубину, чтобы собрать по дереву все индексы вхождений паттерна в текст

Дневник отладки

№	Описание ошибки	Способ устранения
1	WA4: в поле enter структуры ребра в ситуации, когда у текущего ребра есть дети, и при добавлении нового суффикса необходимо выполнить split этого ребра, сохранялся неправильный индекс начала исходного суффикса.	Ошибка произошло из-за невнимательности, для ее устранения я пересмотрела код и поменяла индекс.

Тест производительности



По результатам тестирования видно, что алгоритм Ахо-Корасик поиска подстроки в строке выигрывает по времени у суффиксного дерева, построенного наивным алгоритмом, благодаря использованию префиксного дерева и связей неудач.
Алгоритм построения суффиксного дерева наивным методом соответствует сложности O(n^2), где n — длина текста, тогда сложность поиска вместе с построением составит O(n^2 + m + k), где m - длина паттерна, k - количество вхождений паттерна в текст.

Выводы

В этой лабораторной работе я познакомилась с практическим применением суффиксного дерева. В отличие от других алгоритмов поиска подстроки в строки, суффиксное дерево позволяет обработать текст, а не паттерны. Я научилась строить суффиксное дерево наивным методом и выполнять в нем поиск. Также познакомилась с алгоритмом Укконена, который делает эту структуру данных пожалуй наиболее удобным решением задач поиска множества неизвестных образцов в заранее известном тексте.