Лабораторная работа № 5 по курсу дискретного анализа: Суффиксные деревья

Выполнил студент группы М8О-308Б-21 *Армишев Кирилл.*

Условие

Необходимо реализовать алгоритм Укконена построения суффиксного дерева за линейное время и выполнить свой вариант задания.

Вариант 5: Поиск наибольшей общей подстроки
Найти самую длинную общую подстроку двух строк с использованием суфф. дерева.
Входные данные:
Две строки.
Выходные данные:
На первой строке нужно распечатать длину максимальной общей подстроки, затем перечислить все возможные варианты общих подстрок этой длины в порядке лексикографического возрастания без повторов.

Метод решения

Суффиксное дерево – это *compacttrie*, который построен по всем суффиксам текущей строки, начиная от несобственного суффикса и заканчивая суффиксом, состоящем из одной буквы, обладающий следующими свойствами:

- этот *trie* должен быть сжатый – если есть внутренняя вершина, то у нее должно быть как минимум два потомка
- в нем должно быть такое же количество листьев, как и символов в нашей строке (как и суффиксов в нашей строке)
- не может быть такого, что из 1 вершины выходят два ребра, помеченных строками, начинающимися на одну и ту же букву

Суффиксное дерево можно построить наивным образом за $O(n^2)$, где n – длина текста. Чтобы строить суффиксное дерево максимально быстро – за линейную сложность – существует алгоритм Укконена, в котором есть 3 важных правила:

- добавляем в лист – дописать букву на ребро
- нет пути – создать новый лист
- есть строка – ничего не делаем

Также необходимы важные эвристики, которые ускоряют алгоритм: вместо строк мы на ребрах храним две переменные типа int, обозначающие начало и конец текста на ребре, при этом используя глобальную переменную end и инкрементируя ее на каждой итерации. Помимо этого, есть такое понятие, как суффиксная ссылка, которая перемещает нас из строки вида $x\alpha$ к строке α, где x - символ, а α - строка в уже построенном суффиксном дереве, и, так называемые, прыжки по счетчику, которые помогают нам при переходе между ребрами не проверять несколько символов и перешагивать по ним. Именно все эти эвристики и правила делают алгоритм Укконена позволяют добиться линейной сложности.

Описание программы

Разобьем алгоритм на следующие шаги:

1. Реализация алгоритма Укконена построения суффиксного дерева
   (a) Реализациия класса Node - вершин суффиксного дерева
   (b) Реализация класса SuffixTree - реализация самого суффиксного дерева, который включает в себя следующие функции:
      i. BuildSuffixTree() - основная функция построения, которая пробегает входной текст и для каждого индекса(фазы построения суф. дерева) вызывает функцию расширения дерева.
      ii. ExtendSuffixTree() - расширение суффиксного дерева
      iii. EdgeLength(Node *node) - подсчет длины строки на дуге
2. Реализация алгоритма поиска максимальной подстроки(через обход дерева в глубинку, который реализован в одной функции, выпоняющей два действия:
   (a) Подъем из листьев к корню и проставление у нод пары чисел(x1,x2) - где x1,x2 принимают значение от 0 до 1, и обозначают принадлежность к первой или второй строке(т.е. если подстрока принадлежит обоим строкам, то будет(1, 1).
   (b) Поиск ноды, максимально удаленной от корня(по суммарной длине дуги до этой ноды), у которой стоит(1,1). (1,1) обозначает, что префикс до этой ноды принадлежит обеим строкам.
3. В int main() мы вызываем конструктор SuffixTree, которому передаем строку text1+#+text2. Он добавляет к данной строке $ и вызывает функцию BuildSuffixTree().

Весь код содержится в файле **main.c**: в нем реализованы класссы: Node и SuffixTree. В int main() происходит считывание входных строк, их конкатенация, вызов конструктора SuffixTree, вызов функции обхода дерева с пометкой вершин, вывод максимальной длины общей подстроки и самих максимальных подстрок. Все функции реализованы согласно определениям и алгоритмам, описанным на лекциях.

Дневник отладки

1. 25 сен 2023, 14:44:07 WA на 8 тесте.

   Причина: ошибка в реализации функции, которая помечает ноды на принадлежность строкам. Функция неправильно вычисляла значения у детей корня. После исправления ошибки программа прошла все тесты.

Тест производительности

Сравним написанную программу с наивным поиском максимальной подстроки: Этот алгоритм сравнивает все подстроки первой строки со всеми подстроками второй строки и сохраняет самую длинную общую подстроку. Этот наивный алгоритм будет работать за время $O(n^3)$. Процесс работы этого алгоритма можно описать следующим образом:

1. Генерация всех возможных подстрок для обеих строк (по n^2 подстрок для каждой строки).

2. Сравнение каждой подстроки из первой строки с каждой подстрокой из второй строки.

3. Поиск самой длинной общей подстроки среди всех совпадающих подстрок.

Для сравнения я сгенерировал тесты с различной длиной строк.

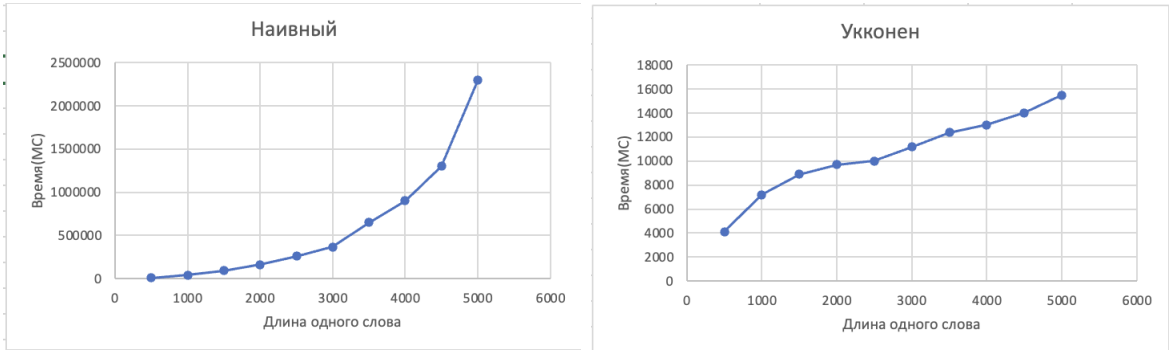

Рис. 1: Сравнение алгоритма Укконена с наивным алгоритмом

Время(мс)	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
Укконен	4105	7211	8936	9711	10002	11164	12384	13263	14211	15500
Наивный	10614	43124	93056	164268	259606	368506	650000	901234	1321001	2310219

Рис. 2: Измерения для графиков

Видно, что наивный алгоритм работает в разы медленнее чем алгоритм Укконена.

Выводы

Проделав данную лабораторную работу, я реализовал алгоритм Укконена построения суффиксного дерева за линейное, отладил его, а затем реализовал алгоритм обхода дерева и поиска наибольшей общей дуги. Суффиксное дерево довольно мощная структура данных, имеющая множество приложений в задачах обработки строк. Среди них нахождение включения одной строки в текст, поиск наибольшей общей подстроки для набора строк, и т.д . Так же суффиксное дерево является вспомогательным, например, при построении суффиксных массивов, поиске статистики совпадений и скорее всего где-нибудь еще. Написание суффиксного дерева оказалось доволно сложным, так как оно содержит множество ивристик, в которых необходимо разбираться.