Лабораторная работа № 5 по курсу дискретного анализа: Суффиксные деревья

Условие

Необходимо реализовать суффиксное дерево. С его помощью требуется линеаризовать циклическую строку, то есть найти минимальный в лексикографическом смысле разрез циклической строки.

Метод решения

Требуется реализовать алгоритм Укконена построения суффиксного дерева за линейное время. Алгоритм Укконена в самой простоей реализации имеет сложность $O(n^3)$, так как добавляет каждый суффикс каждого префикса строки в отличии от добавления всех суффиксов строки. Основная идея в том, чтобы оптимизировать его и получить сложность $O(n)$.

Суффиксные ссылки позволяют не проходить каждый раз по дереву из корня. Для построения суффиксных ссылок достаточно хранить номер последней созданной вершины при продлении. Если на этой же фазе мы создаём ещё одну новую вершину, то нужно построить суффиксную ссылку из предыдущей в текущую. Сложность алгоритма с использованием продления суффиксов и суффиксных ссылок $O(n^2)$.

Для ускорения до $O(n)$ нужно уменьшить объём потребляемой памяти. Будем в каждом ребре дерева хранить не подстроку, а только индекс начала и конца подстроки.

Для нахождения минимального лексикографического разреза строки s построим суффиксное дерево от строки $s + s$ и найдём лексикографически минимальный путь длины $|s|$ в дереве.

Временная и пространственная сложность решения будет $O(n)$.

Описание программ

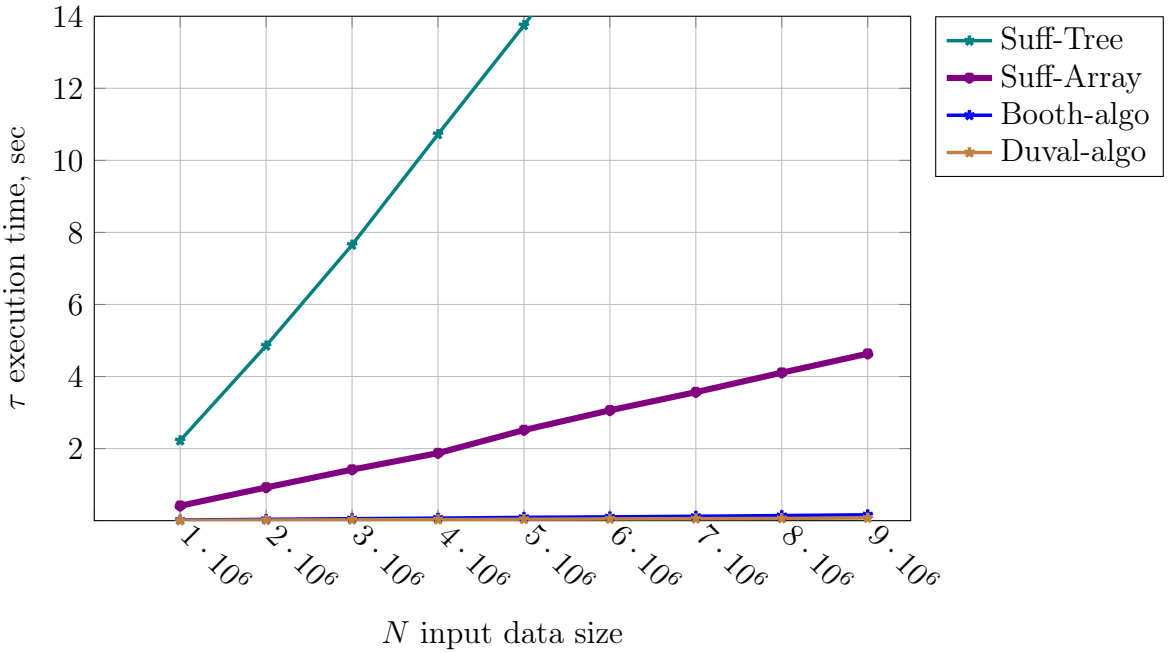
К сожалению, тестирующая система не позволила использовать модульный подход к решению поставленной задачи, поэтому весь рабочий код размещён в единственном файле *main.cpp*.

Основным классом является *TSuffTree*. Его конструктор выполняет алгоритм Укконена построения дерева. Для этого используются вспомогательные структуры:

- TNode — вершина дерева. Содержит суффиксну ссылку и ребра до своих сыновей.
- TEdge — ребро дерева. Содержит конечную вершину ребра, и два числа определяющие какая подстрока текста соответствует данному ребру.
- TState — текущее положение в дереве. Используется для итерации по дереву, в том числе и при построении.

После реализации алгоритма Укконена, остаётся добавить одну единственную функцию *lexic_min_cut*. Она будет спускаться по дереву на глубину $|s|$, при каждой возможности выбирая лексикографически наименьшее ребро (отметим, что сентинел в данном случае является максимальным). Сама функция выглядит по-разному в зависимости от структуры хранящей ребра в вершине. В случае *std::map* просто достаёт самое первое ребро, а в случае *std::unordered_map* — по-честному по всем рёбрам выбираем минимум на каждом шагу.

Тест производительности



Выводы

Во время выполнения лабораторной работы вспомнил суффиксные структуры данных, способы хранения графов, реализовал алгоритм Укконена, ознакомился с приложениями суффиксного дерева, изучил различные способы построение суффиксного массива и их особенности. Суффиксное дерево, как и массив, позволяет быстро искать множество шаблонов в тексте, что значительно отличает его от остальных алгоритмов поиска вхождений: Кнута-Морриса-Пратта, Бойера-Мура. Часто суффиксное дерево является бесполезным инструментом, так как оно обладает целым рядом минусов: слишком большой расход памяти, большая константа в линейной асимптотике его построения, сложность написания кода и др. Бывает, что в задачах, где может предполагаться решение с использованием суффиксного дерева, можно обойтись суффиксным массивом (для построение которого можно использовать линейные алгоритмы, не использующие дерево DC3, SA-IS), что уже может ускорить программу, не говоря даже о возможном существовании других методов решения. Так, например, задачу моего варианта можно решить, как минимум 5 другими способами, 2 из которых тоже имеют асимптотику $O(n)$, но работают на порядок быстрее. Этого не видно из графика, но на входной строке порядка 10^7 символов самое эффективное решение отработало в 338 раз быстрее, чем решение с суффиксным деревом, и использовало $O(1)$ дополнительной памяти!

Таким образом, суффиксное дерево может решать множество задач, а эффективный алгоритм Укконена, позволяющий строить дерево за линейное время, также стимулирует использование этой структуры повсеместно. Однако стоит помнить, что не существует универсальных инструментов, которые решают задачу оптимально.