Лабораторная работа № 5 по курсу дискретного анализа: Суффиксные деревья

Выполнил студент группы М8О-308Б-22 МАИ *Немкова Анастасия.*

Условие

Найти в заранее известном тексте поступающие на вход образцы.
Задача: 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.
Вариант:
1. Поиск в известном тексте неизвестных заранее образцов

Метод решения

К исходному тексту добавляем терминальный символ и строим суффиксное дерево по алгоритму Укконена. Основные правила, которые используются при добавлении в дерево нового символа:

1) После добавления символа в дерево, активная вершина (active node) устанавливается на корень, а активное ребро (active edge) указывает на первый символ нового суффикса. Если активная длина (active len) равна нулю, активное ребро устанавливается на текущую позицию символа.

2) Если активное ребро разделяется, создается новая вершина. Если это не первая вставленная вершина на текущем шаге, устанавливается суффиксная ссылка (suffix link) от ранее вставленной вершины к новой.

3) Если активная вершина не является корнем и после разделения рёбер у нас есть суффиксная ссылка, активная вершина переходит к вершине, на которую указывает эта ссылка. В противном случае активная вершина возвращается к корню.

При поиске образца начинаем с корня, и для каждого символа искомой строки проверяется наличие дочерних узлов в текущей вершине. Если дочерний узел найден, выполняется проход по символам ребра, соответствующего этому узлу. Если все символы совпадают, продолжается поиск, и в случае достижения конца искомой строки из текущей вершины обходятся все листья и выводятся их номера в порядке возрастания.

Описание программы

Было реализовано суффиксное дерево, каждый узел которого содержит:

- std::map<char, TNode*> children - переходы к дочерним узлам
- TNode* suffixLink - суффиксная ссылка
- int start - индекс начала образца
- int* end - указатель на индекс конца образца
- int suffInd - позиция суффикса, если узел соответствует концу суффикса; -1, если нет

Сам класс TSuffixTree содержит:

- std::string text - текст, из которого строится суффиксное дерево
- TNode* root - указатель на корневой узел дерева
- TNode* activeNode - указатель на активный узел, в котором происходит вставка символов
- TNode* lastAddNode - указатель на последний добавленный узел для обновления суффиксной ссылки
- int activeEdge - индекс первого символа ребра по которому мы будем спускаться
- int activeLen - количество символов, которое мы прошли по ребру
- int remainder - количество суффиксов которые осталось добавить
- int leafEnd - конечный индекс для листа

В данном классе реализованы методы:

- std::vector<int> Search(const std::string pattern) - поиск всех вхождений паттерна в тексте
- void CountIndex(TNode* node, std::vector<int> v) - рекурсивный обход все узлов дерева и подсчет индексов листьевб выходящих их данной вершины
- int EdgeLen(TNode* node) - длина ребра для данного узла
- bool GoDown(TNode* node) - проход вниз по дереву
- void InsertCharacter(int pos) - вставка символа из текста в суффиксное дерево
- void Destroy(TNode* node) - удаление корня и всех дочерних узлов

Дневник отладки

1. 25 сен 2024, 16:16:07 WA на 3 тесте

   Для образцов, не найденных в тексте, выводился номер образца Решение: для начала проверяем что вектор с индексами вхождений не пуст, а потом выводим индекс образца

Тест производительности

Для измерения производительсти сравнивается время поиска в реализованном суффиксном дереве и с использованием std::string.find(). Подсчет времени производился с помощью библиотеки chrono, которая позволяет фиксировать время в начале и конце выполнения сортировки. Первый тест состоит из текста длиной 10^4 символов, второй из $5*10^4$ символов, третий из 10^5, четвертый из $5*10^5$. Количество паттернов в каждом тесте - 100, их длина 50 символов

На графике представлена зависимости времени выполнения поиска паттернов от объёма входных данных. Сложность std::string.find() - O(n * m).

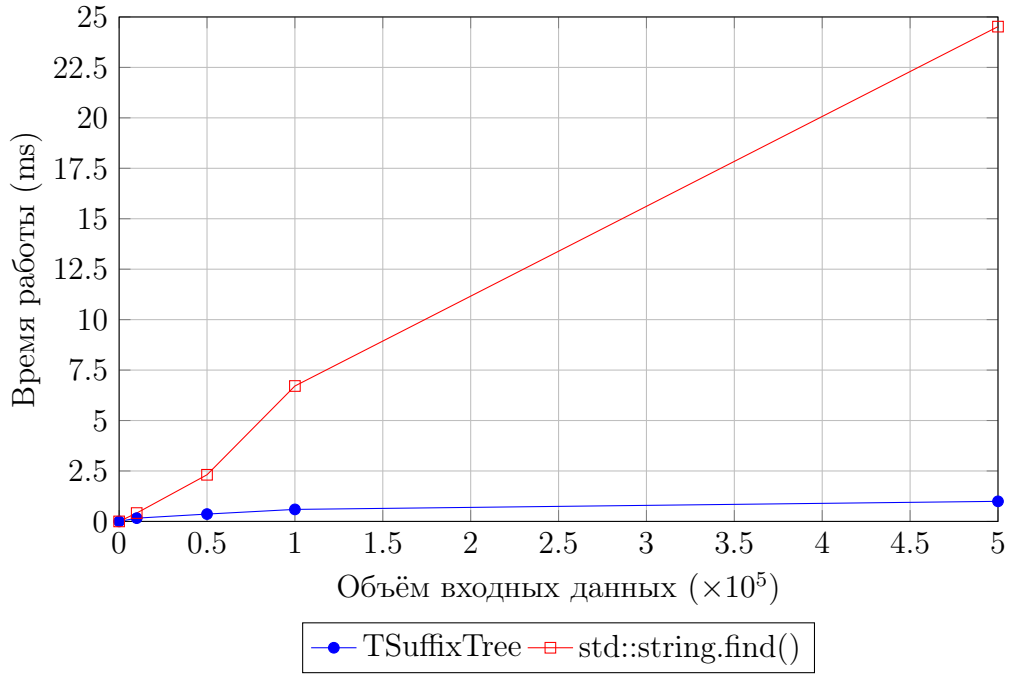

Рис. 1: Сравнение времени выполнения поиска суффиксным деревом и std::string.find()

Выводы

В ходе данной лабораторной работе было реализовано суффиксное дерево с использованием алгоритма Укконена, что позволило добиться линейной сложности построения

дерева и эффективного поиска подстрок. Суффиксное дерево обеспечивает поиск подстрок за время, пропорциональное длине паттерна, что значительно быстрее по сравнению с методом std::string::find, сложность которого в худшем случае составляет O(n * m). Это делает суффиксное дерево более предпочтительным для задач множественного поиска подстрок в больших текстах.