Лабораторная работа №5 по курсу дискретного анализа: Суффиксные деревья

Выполнил студент группы М8О-309Б-23 Кривошапкин Егор

Условие

Найти в заранее известном тексте поступающие на вход образцы с использованием суффиксного массива. 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

Решение задачи можно разделить на два шага:

1) Построение суффиксного массива.
Чтобы составить суффиксный массив, в конец строки сначала добавляется специальный символ, который гарантирует, что все суффиксы будут различаться. Сначала суффиксы сортируются только по первому символу. Затем применяется итеративный метод: на каждом шаге упорядочиваем суффиксы уже по первым 2^k символам (для ускорения используется поразрядная сортировка). Параллельно каждому суффиксу присваивается номер его класса эквивалентности — одинаковый для совпадающих подстрок длины 2^k. Когда все классы становятся уникальными, работа алгоритма заканчивается, и на выходе получаем готовый суффиксный массив.

2) Поиск подстроки с помощью суффиксного массива.
Здесь применяется бинарный поиск. Сначала находим позицию первого суффикса, который начинается с искомого образца (или ближайшего большего по алфавиту, если точного совпадения нет). Затем ищем границу справа — первый суффикс, который строго больше образца. Разница между этими двумя индексами показывает, сколько раз образец встречается в тексте. Поскольку суффиксный массив хранит позиции всех суффиксов в исходной строке, мы сразу можем узнать, где именно находятся вхождения искомого шаблона.

Описание программы

Программа состоит из нескольких функций:
- **`countingSort`** – универсальная реализация сортировки подсчётом, которая используется при построении суффиксного массива.

- **`comparePattern`** – вспомогательная функция, которая сравнивает подстроку текста, начинающуюся с позиции `pos`, с заданным образцом.

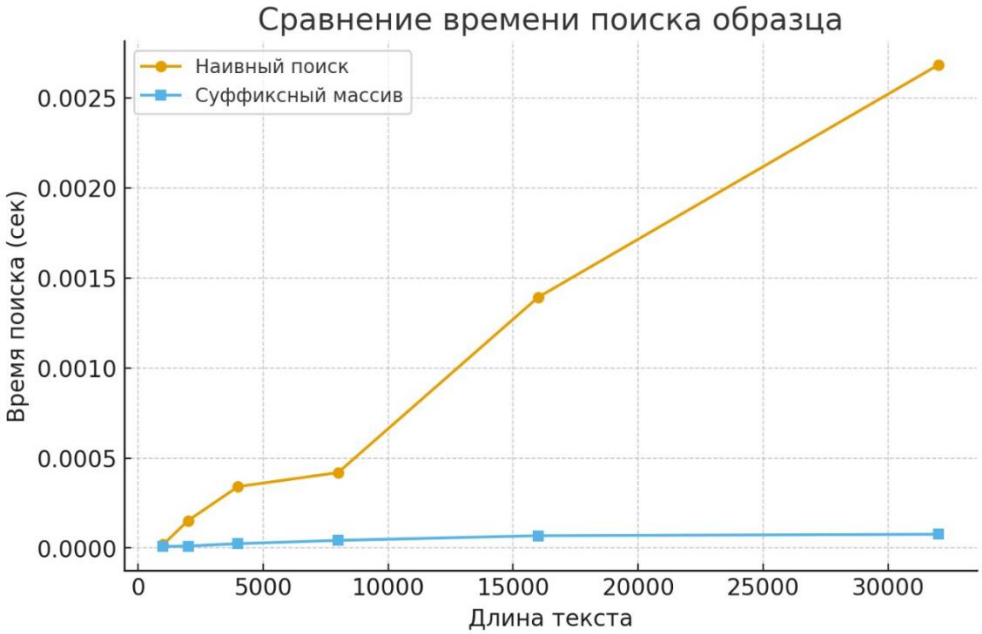
- **`findRange`** – выполняет бинарный поиск по суффиксному массиву, чтобы найти диапазон вхождений образца в тексте. Возвращает границы этого диапазона.

- **`main`** – основная функция программы.

- Считывает исходный текст.
- Строит для него суффиксный массив.
- Далее построчно считывает образцы для поиска.
- Для каждого образца находит диапазон вхождений с помощью `findRange`.
- Если вхождения есть, выводит их позиции

Тест производительности

На графике видно, что среднее время наивного поиска увеличивается почти линейно с ростом длины текста. В то же время поиск по суффиксному массиву остаётся практически неизменным, так как работает за O(mlogn), где m — длина шаблона.
Как видим, алгоритм поиска при помощи суффиксного массива в десятки раз выигрывает по скорости работы у наивного алгоритма при большом количестве поступающих на сравнение образцов. Что неудивительно, ведь в алгоритме применяется разовая обработка текста, которая позволяет удобно применить бинарный поиск для поиска вхождений образцов.



Сложность алгоритма:

1. **Построение суффиксного массива.**
   Сначала суффиксы сортируются по первым символам с помощью std::ranges::sort, что занимает O(n*logn). Затем на каждом шаге выполняется сортировка по классам эквивалентности при помощи сортировки подсчетом (countingSort). Поскольку классов эквивалентности не больше n, каждая такая сортировка работает за O(n). Таких шагов порядка logn, так как на каждой итерации длина сравниваемых подстрок удваивается. Итоговая сложность построения суффиксного массива равна O(nlogn).
2. **Поиск образцов.**
   Для поиска используется бинарный поиск по суффиксному массиву (findRange), работающий за O(logn). При этом сравнение суффикса с образцом (comparePattern) может потребовать проверки до m символов. Если искать q различных образцов, общая сложность поиска составит O(q*m*logn).
3. **Вывод результатов.**
   После нахождения вхождений программа сортирует их индексы при помощи std::sort. Если число вхождений обозначить как k, то сортировка занимает O(k*logk). В худшем случае сортировка выполняется для каждого из q образцов, и тогда итоговая сложность этого этапа равна O(qk*logk).

Итоговая сложность работы программы:
O(n*logn* + *q*(m*logn* + *k*logk)),
где n – длина текста, m – длина самого длинного образца, q – число образцов, k – максимальное число вхождений одного образца.

Выводы

В результате проведения лабораторного исследования был разработан алгоритм создания суффиксного массива для нахождения строк в тексте. Проведенные испытания выявили, что простой метод поиска демонстрирует значительное снижение скорости работы с ростом размера текстовых данных. В то же время, применение суффиксного массива обеспечивает многократное ускорение процесса поиска, что становится особенно заметно при обработке множества запросов. Несмотря на требование дополнительных ресурсов для своего формирования, суффиксный массив полностью оправдывает эти затраты в условиях многократного использования. Следовательно, практическая реализация наглядно подтвердила преимущества алгоритмов с улучшенной асимптотикой перед простыми, но плохо масштабируемыми решениями.