Суффиксные деревья

Выполнил студент группы 08-208 МАИ *Ибрагимов Далгат.*

Постановка задачи

Вариант 4: Линеаризация циклической строки

Необходимо найти минимальную лексикографическую циклическую перестановку строки. Для решения этой задачи рассмотрим и сравним два алгоритма:

1. Суффиксное дерево.
2. Алгоритм Бо Лэнга.

Алгоритм 1: Суффиксное дерево

Описание: Суффиксное дерево позволяет эффективно работать с различными строковыми задачами, такими как поиск подстрок, сравнение строк и нахождение минимальной циклической перестановки. Суффиксное дерево — это дерево, которое содержит все суффиксы строки как свои подстроки.

Этапы работы алгоритма:

1. Строим суффиксное дерево для строки.
2. Добавляем строку к самой себе для учета циклических сдвигов.
3. Поиск минимальной циклической перестановки с помощью обхода дерева.

Сложность алгоритма: Построение суффиксного дерева имеет временную сложность $O(n)$, где n — длина строки. Поиск минимальной циклической перестановки также требует $O(n)$ операций.

Метод решения

Моя программа реализует суффиксное дерево для поиска минимальной циклической перестановки строки. Алгоритм построения суффиксного дерева основан на методе Укконена, который строит дерево за линейное время $O(n)$ по длине строки. Этот алгоритм позволяет эффективно решать задачи поиска и сравнения строк.

Программа состоит из двух основных этапов:

1. Построение суффиксного дерева для удвоенной строки.
2. Поиск минимальной циклической перестановки путем сравнения суффиксов.

Описание программы

Для решения задачи были разработаны следующие классы и методы:

- **struct SuffixTreeNode** — структура для представления узла суффиксного дерева. Узел содержит:
  - `Children` — контейнер для хранения потомков узла (ассоциативный массив).
  - `Start, End` — границы подстроки, соответствующей этому узлу.
  - `SuffixLink` — суффиксная ссылка для быстрой навигации между узлами.
  - `SuffixIndex` — индекс суффикса, если узел является листом дерева.
  - `IsDynamicEnd` — флаг, указывающий, был ли `End` динамически выделен.
- **class SuffixTree** — класс для построения и работы с суффиксным деревом:
  - `SuffixTree(const std::string& s)` — конструктор, который строит суффиксное дерево по строке `s` с использованием алгоритма Укконена. Удваивает строку для учета циклических сдвигов.
  - `std::string GetMinimalCyclicShift()` — функция, которая находит минимальную циклическую перестановку строки, используя суффиксное дерево. Этот метод работает, сравнивая индексы суффиксов строки.

Описание функций

- `NewNode(int start, int* end, bool isDynamic = false)` — создает новый узел дерева с указанием диапазона суффикса (`start`, `end`) и флага `isDynamic`, указывающего, выделен ли `end` динамически.
- `ExtendSuffixTree(int pos)` — основной метод для построения дерева. Он отвечает за добавление новых суффиксов в дерево и поддержание его структуры с суффиксными ссылками. Этот метод реализует правила построения дерева Укконена.
- `WalkDown(SuffixTreeNode* currNode)` — функция для перемещения вниз по дереву. Она используется для оптимизации построения дерева и быстрого перемещения между узлами.
- `SetSuffixIndexByDFS(SuffixTreeNode* n, int labelHeight)` — функция для установки индексов суффиксов при обходе дерева в глубину (DFS). Используется для маркировки листовых узлов дерева.
- `EdgeLength(SuffixTreeNode* n)` — возвращает длину ребра между текущим узлом и его потомком, что помогает при навигации по дереву.

Дневник отладки

- Во время разработки возникла проблема с корректной обработкой строк в процессе сравнения суффиксов. Было принято решение перейти на работу с индексами вместо создания новых строк, что значительно уменьшило использование памяти.
- Алгоритм Укконена при построении дерева потребовал тщательной отладки суффиксных ссылок, поскольку ошибки в их установке приводили к неправильному построению дерева.
- Также возникли сложности с управлением памятью, так как узлы дерева используют динамически выделяемую память для хранения концов суффиксов. Для решения этой проблемы были добавлены флаги для указания динамического выделения, что позволило избежать ошибок при освобождении памяти.

Использованный алгоритм

Для построения суффиксного дерева использован алгоритм Укконена, который работает за $O(n)$ и является одним из самых эффективных методов построения суффиксных деревьев. Этот алгоритм позволяет строить суффиксное дерево на лету, не требуя заранее знать всю строку.

Линеаризация циклической строки осуществляется путём поиска минимальной суффиксной перестановки на основе сравнения суффиксов строки. В результате программа находит индекс, с которого начинается минимальная лексикографическая перестановка.

Тестирование производительности

Сравним с другим алгоритмом. Простейшая реализация алгоритма Бо Лэнга решает задачу поиска минимальной лексикографической циклической перестановки строки за время $O(n)$.

Для оценки производительности были проведены тесты с разными длинами строк. На графике ниже показаны времена выполнения суффиксного дерева и алгоритма Бо Лэнга в зависимости от длины строки.

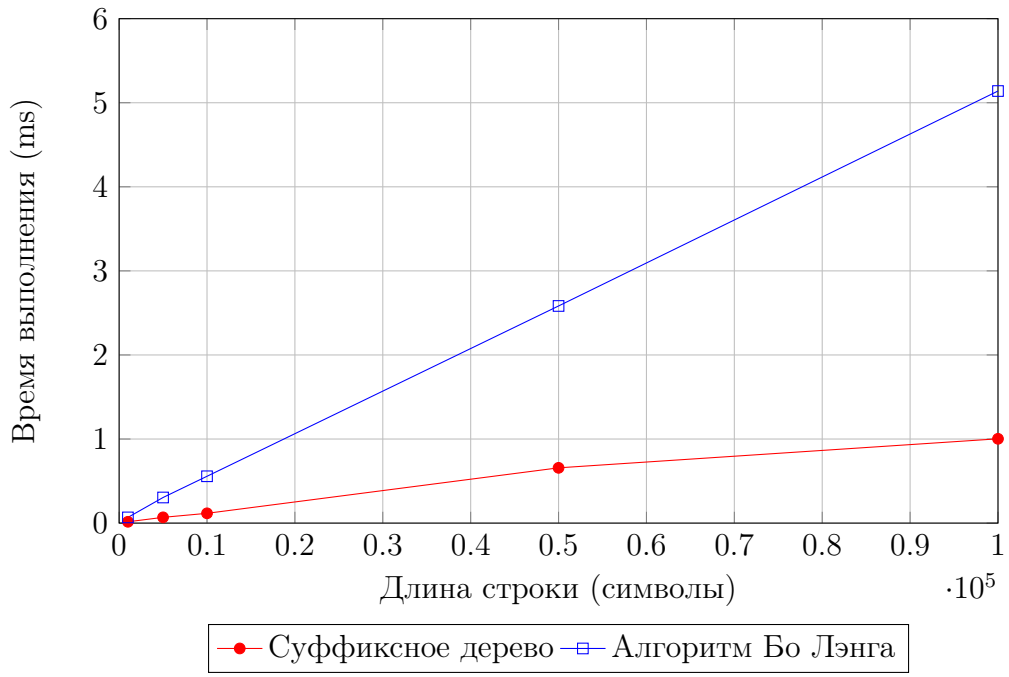

Рис. 1: Сравнение времени выполнения суффиксного дерева и алгоритма Бо Лэнга

Выводы

Суффиксное дерево обладает большей гибкостью и может решать более широкий круг задач, таких как поиск подстрок или нахождение общих подстрок. Однако для задачи минимальной циклической перестановки алгоритм Бо Лэнга должен работать быстрее и эффективнее, но из-за плохой наивной реализации этого не происходит