Лабораторная работа № 2 по курсу дискретного анализа: Сбалансированные деревья

Выполнил студент группы 08-215 МАИ *Тараскаев Давид*.

Условие

Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до 2^{64} - 1.
Структура данных: AVL-дерево.

1. Представление ключей: при каждом вводе ключ приводится к нижнему регистру функцией `to_lower`.
2. Узлы дерева: структура `Node` хранит:
   - `char key[257]` — ключ-строку;
   - `uint64_t value` — связанное значение;
   - `int height` — высоту поддерева;
   - указатели на левого и правого потомка.
3. Балансировка: при вставке и удалении вычисляется баланс-фактор, при необходимости делаются одно- и двукратные повороты (`rotateLeft`, `rotateRight`, `bigRotateLeft`, `bigRotateRight`).
4. Сохранение/загрузка: `SaveToFile` рекурсивно обходит узлы, записывая маркер присутствия, ключ и значение; `LoadFromFile` восстанавливает дерево по тем же правилам, проверяя целостность формата.

Описание программы

to_lower(src, dst) — копирует строку `src` в `dst`, переводя буквы в нижний регистр.

`struct Node` — представление узла AVL-дерева с полями `key`, `value`, `height`, `left`, `right`.

`class AVL_Tree` — класс-обёртка над корнем дерева, даёт методы:

- `Insert(key, value)` — вставка новой пары (или «Exist»);
- `Remove(key)` — удаление узла по ключу (или «NoSuchWord»);

- `Find(key)` — поиск значения (или «NoSuchWord»);
- `InorderPrint()` — вывести все значения в порядке возрастания ключей;
- `SaveToFile(path)`, `LoadFromFile(path,...)`.
- `Повороты и балансировка` реализованы приватными методами `rotateLeft/Right` и `bigRotateLeft/Right`.

Дневник отладки

Изначально написал код, ни чем себя не ограничивая, потом заменял запрещенные структуры на разрешенные. WA получал из-за неправильного отлавливания ошибок в ходе работы программы.

Тест производительности

В реализованном AVL дереве 10 миллионов случайных комманд выполняется за 28.52 секунд, а в встроенной в stl map - 28.74.
Результат нормальный, так как map - это реализация красно-черного дерева и сложность методов поиска, вставки и удаления у обоих деревьев одинаковая - O(log n)

Выводы

Так как средняя временная сложность алгоритма Бойера-Мура O(N), он исключительно эффективен для задач поиска последовательностей в больших объемах текстовых данных. Это делает его предпочтительным выбором для таких приложений, как функции поиска в текстовых редакторах, анализ исходного кода, поиск специфических последовательностей в биоинформатике, обнаружение известных сигнатур в системах компьютерной безопасности и быстрая фильтрация данных в объемных лог-файлах