

Лабораторная работа №2 по курсу дискретного анализа: Сбалансированные деревья

Выполнил студент группы М8О-212Б-22 МАИ *Юрков Евгений*.

Условие

Вариант: 2

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 264 - 1. Разным словам может быть поставлен в соответствие один и тот же номер.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

- **+ word 34** — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.
- **- word** — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.
- **word** — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».
- **! Save /path/to/file** — сохранить словарь в бинарном компактном представлении на диск в файл, указанный параметром команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже).
- **! Load /path/to/file** — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

Различия вариантов заключаются только в используемых структурах данных: **Красно-чёрное дерево**.

Метод решения

Была реализована структура данных `map` на основе красно-черного дерева. Эта структура данных предоставляет функционал для вставки, удаления и поиска данных. Также были реализованы функции сохранения дерева в файл и загрузки дерева из файла.

Описание программы

Класс `rb_tree` содержит в себе структуру узла дерева и реализует сбалансированное дерево поиска. Поддерживает операции вставки, удаления и поиска.

Класс `map` является оболочкой вокруг красно-черного дерева. Предоставляет часть интерфейса `textttstd::map`.

Методы `save`, `load` являются специализированными для `map<std::string, unsigned long long>`.

Дневник отладки

1. Программа успешно прошла все тесты, в которых не было функций загрузки и сохранения.
2. Программа получила ML на тесте 7. Проблема была в том, что при сохранении и загрузке дерева данные сначала записывались в вектор, а потом в файл.
3. Программа получила TL на тесте 7. Проблема была в способе хранения дерева в файле: дерево строилось и балансировалось каждый раз заново, так как использовалась функция `insert`. Нужно было сохранять дерево именно в том виде в котором оно было, запоминая правого и левого потомка.
4. Программа получила WA на тесте 9. Команда `save` была написана с маленькой буквы.
5. После исправления регистра команды `Save`, программа прошла все тесты.

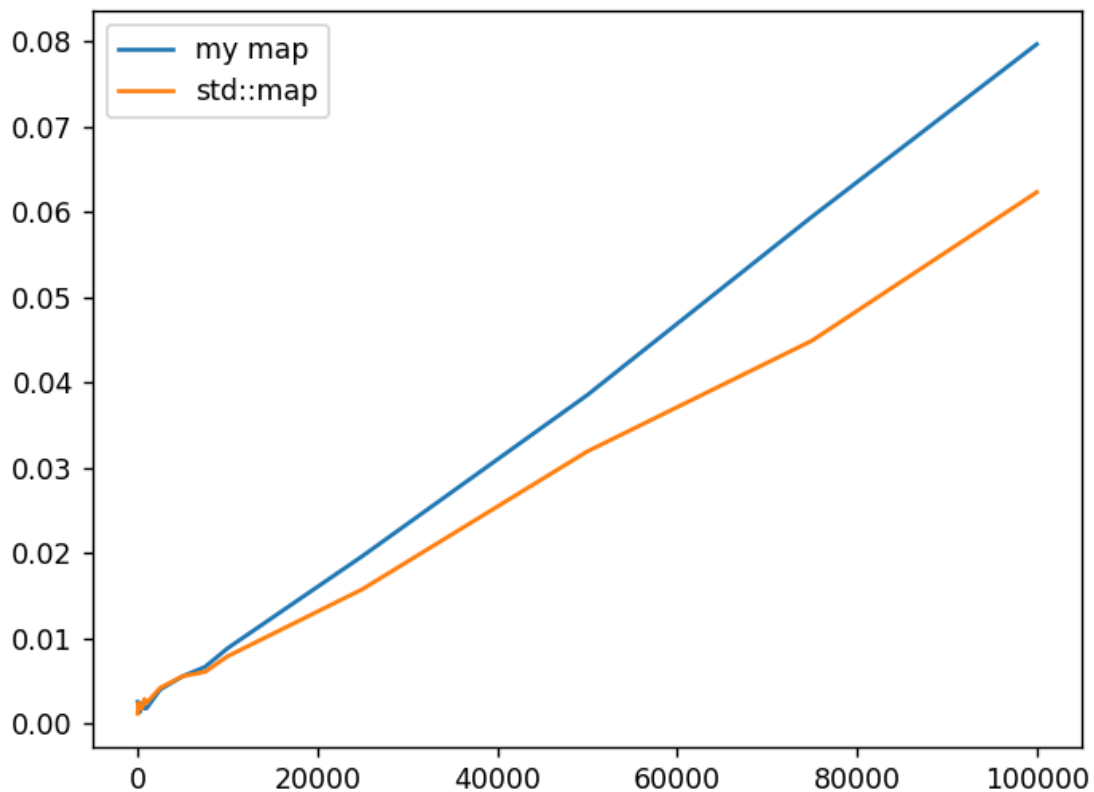


Рис. 1: График зависимости времени работы программы от количества введенных данных

Тест производительности

Сложность вставки, поиска и удаления элементов в дереве $O(\log n)$, где n - количество элементов дерева. Для построения графика (Рис. 1) использовались тесты с 1 - 100000 строками данных. Из графика видно, что время работы сравнимо с временем работы `std::map`.

Выводы

Сбалансированные деревья играют важную роль в хранении и поиске данных. Они обеспечивают логарифмическое время выполнения операций вставки, поиска и удаления, что делает их особенно полезными в задачах, где необходимо обеспечить быстрый доступ к данным, например, в построении эффективных баз данных и ассоциативных массивов. Однако недостатком таких деревьев является сложность реализации балансировки дерева для операций вставки и удаления элементов.