Лабораторная работа № 2 по курсу дискретного анализа: сбалансированные деревья

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

Условие

Общая постановка задачи: 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 $2^{64} - 1$. 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. — 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 парамером 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.

Вариант структуры данных: Декартово дерево

Метод решения

Моя программа реализует структуру данных Treap, которая объединяет свойства двоичного дерева поиска (BST) и кучи (heap). Treap используется для хранения пар ключ-значение и поддерживает быстрое выполнение операций вставки, удаления и поиска. Программа обрабатывает команды из входного файла до его окончания, разделяя ввод на команду, ключ и значение. В зависимости от команды выполняются соответствующие действия:

Если команда начинается с +, программа считывает ключ и значение, проверяет наличие ключа в дереве, и если его нет, добавляет новую пару в дерево. Если ключ уже существует, выводится соответствующее сообщение. Если команда начинается с - , программа удаляет элемент с указанным ключом из дерева. При вводе ключа без команды программа осуществляет поиск этого ключа в дереве и выводит значение, если ключ найден. Команды ! Save и ! Load выполняют сохранение и загрузку дерева из бинарного файла. Treap поддерживает основное свойство кучи: для любого узла приоритет этого узла больше приоритета его потомков. Это свойство сохраняется при всех операциях над деревом, таких как вставка и удаление. Для этого реализованы следующие функции:

- Node* insert(Node* root, Node* node) — вставка узла с балансировкой приоритетов.
- Node* erase(Node* root, const char* key) — удаление узла с поддержанием свойств кучи.

Описание программы

Для выполнения задания был разработан один класс и одна структура:

- struct Node — структура для представления узла дерева Treap. Узел содержит ключ, значение, приоритет и указатели на левые и правые поддеревья.
- class Treap — реализация Treap, включающая методы для добавления, удаления, поиска узлов, а также для сохранения и загрузки дерева в/из бинарного файла.

Основные методы класса Treap:

- const char* add(const char* word, uint64_t number) — добавление новой пары ключ-значение в дерево.
- const char* del(const char* word) — удаление пары ключ-значение из дерева.
- const char* search(const char* word) — поиск значения по ключу.
- const char* save(const char* filename) — сохранение текущего состояния дерева в бинарный файл.

- const char* load(const char* filename) — загрузка дерева из бинарного файла.

Программа использует структуру данных Treap для хранения словаря, где ключами являются строки, а значениями — числа.

Дневник отладки

Во время разработки возникла проблема с корректностью чтения и записи данных в бинарный файл. Проблема заключалась в неверной обработке формата данных при сохранении и загрузке, что приводило к ошибкам при чтении файла. Для устранения этой проблемы был перепроверен порядок записи и чтения данных, а также учтены возможные ошибки при работе с бинарными файлами.

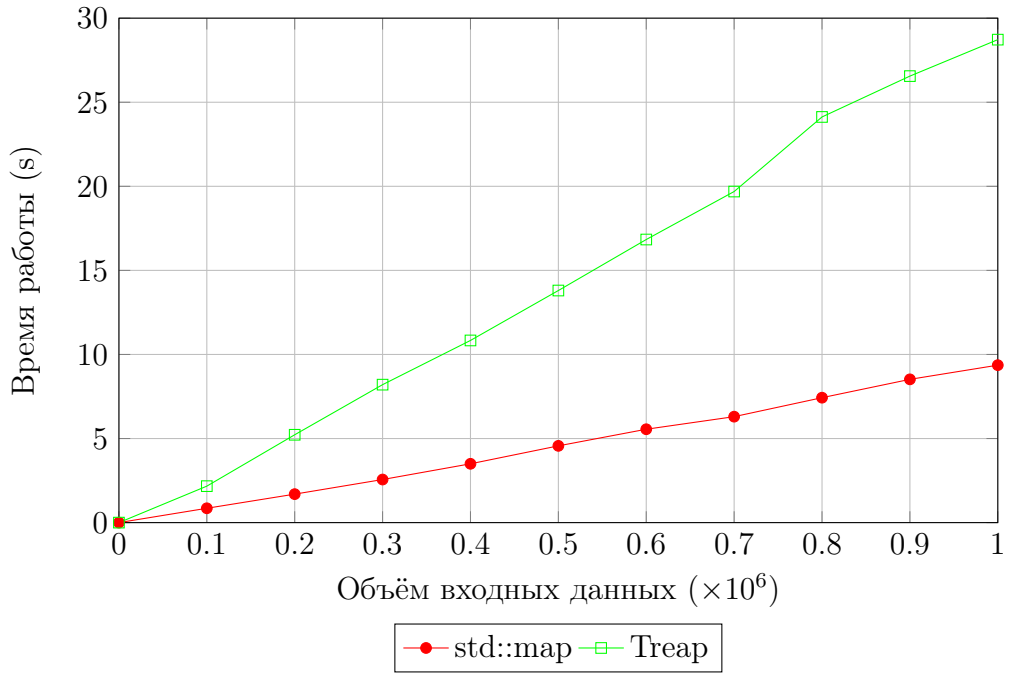
Еще одной сложностью стало управление памятью для строковых данных. В процессе разработки были учтены особенности динамического выделения и освобождения памяти для строк, что позволило избежать утечек памяти.

Тест производительности

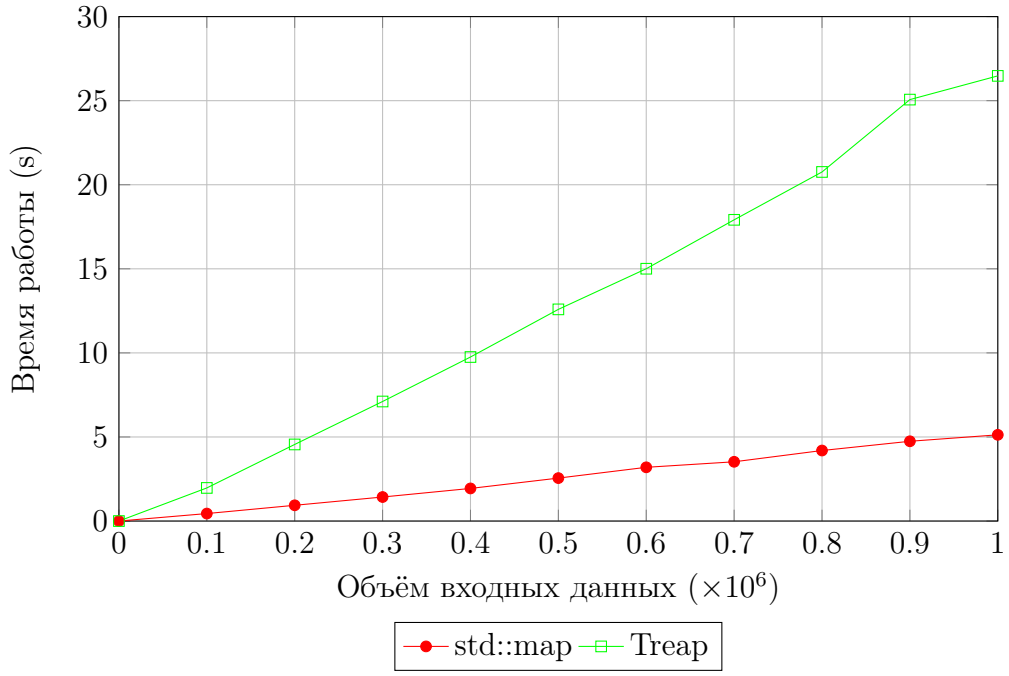
Для проверки производительности моего словаря в виде AVL-дерева я использовала сравнение со стандартным контейнером std::map. Это имеет смысл так как в его основе лежит красно-черное дерево, которое тоже является сбалансированным. Сравнение производилось на входных данных больших размеров, не превышающих 10^5.

Исходя из графика, представленного ниже, можно увидеть, что вставка (аналогично с удалением) элементов в std::map работает быстрее, чем в AVL-дереве, а с поиском ситуация противоположна. Это происходит в силу различий в алгоритмах балансировки: в красно-черных деревьях она происходит гораздо реже, чем в AVL-дереве. Как следствие вставка (удаление) происходит быстрее. Однако высота красно-черного дерева больше высоты соответствующего AVL-дерева. Отсюда вытекает большее время поиска. Не логарифмическая зависимость на графиках, по моему мнению, получается из-за довольно частых выделений/освобождений памяти на кучи, а это, как известно, не дешевая в плане времени операция. К тому же графики имеют очень похожую форму, что говорит об асимптотически одинаковой скорости роста времени выполнения операций.

Вставка



Поиск



Выводы

Реализованный класс Treap позволяет эффективно управлять словарем с поддержкой операций вставки, удаления и поиска. Использование структуры данных Treap обеспечивает сбалансированное выполнение этих операций с учетом свойств двоичного дерева поиска и кучи. Программа успешно сохраняет и восстанавливает состояние дерева из бинарного файла, что делает её удобной для работы с большими объемами данных. std::map показывает лучшую производительность при вставке по сравнению с Treap. std::map основан на красно-чёрных деревьях, которые обеспечивают балансировку с меньшими накладными расходами, что приводит к более эффективной вставке данных. Treap, с другой стороны, имеет дополнительные накладные расходы, связанные с поддержанием структуры приоритета, что усложняет вставку и увеличивает её время. std::map демонстрирует значительно лучшую производительность при выполнении поиска, что связано с эффективностью его балансировки и минимальными накладными расходами на поддержание структуры дерева. Treap, напротив, требует больше времени для поиска из-за дополнительных операций, связанных с поддержанием приоритетов и случайных балансировок, что увеличивает сложность поиска.