Лабораторная работа № 2 по курсу дискртного анализа: сбалансированные деревья

Выполнил студент группы 08-208 МАИ *Армишев Кирилл.*

Условие

Кратко описывается задача:

1. 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 $2^{64} - 1$. При этом разным словам может быть поставлен в соответствие один и тот же номер.

2. Вариант задания: 1. Реализация с использованием АВЛ-дерева.

Метод решения

АВЛ-дерево – это бинарное дерево поиска, близкое к идеально сбалансированному из-за того, что для любого узла выполнятся условие: модуль разности высот левого и правого поддеревьев не больше единицы.

Для того чтобы дерево было сбалансированым его нужно балансировать. В АВЛ-дереве это происходит на основе баланса каждого узла.

Баланс – это разница высот левого и правого педдеревьев этого узла. Возможны два пути: хранить сам баланс-фактор либо хранить высоты поддеревьев и вычислять баланс-фактор по мере надобности.

Для балансировки применяется две вспомогательных процедуры: левый и правый повороты дерева.

Процедура вставки такая же как и в обычном бинарном дереве, но после нее происходит процедура балансировки.

Удаление как и в обычном бинарном дереве, но далее следует балансировка дерева.

Поиск аналогичен поиску в бинарном дереве.

Далее реализованное АВЛ-дерево используется в качетве словаря, над которым производятся операции добавления, удаления, поиска, записи в файл или чтения из файла в зависимости от команд пользователя.

Словарь хранит пары ключ-значение. Ключом выступает строка (не более 256 значащих символов), значением беззнаковое 8-ми байтовое целое(unsigned long long).

Описание программы

Весь код содержится в файле **main.c**: в нем реализован класс АВЛ-дерева, в котором реализованы методы вставки, удаления, поиска, методы для балансировки, поиска

наименьшего элемента и т д. Также реализованы сериализация и десериализация. Все функции реализованы согласно определениям и алгоритмам, описанным на лекциях.

Дневник отладки

1. 25.04 23:32:46 ML на 15 тесте.

   Причина: в рекурсивной функции _addNode выделял лишнюю не нужную память для хранения ключа. Решение: переместил выделение памяти под ключ в функцию addNode, где выделение памяти под новый ключ происходит единственный раз.

Тест производительности

Я выполнил сравнение своей реализации АВЛ-дерева со стандартным контейнером `std::map`. В его основе лежит красно-черное дерево, которое тоже является сбалансированным. Замеры производительности проводились на операциях вставки/поиска/удаления.

Из графиков видно, что, операции вставки и удаления работают медленне, чем `std::map`, а поиск быстрее. Это происходит в силу различий в алгоритмах балансировки: в красно-черных деревьях она происходит гораздо реже, чем в АВЛ. Как следствие вставка и удаление происходит быстрее, но при этом и высота красно-черного дерева больше высоты соответствующего АВЛ дерева. Отсюда вытекает большее время поиска.

Графики имеют очень похожую форму, что говорит об асимптотически одинаковой скорости роста времени выполнения операций

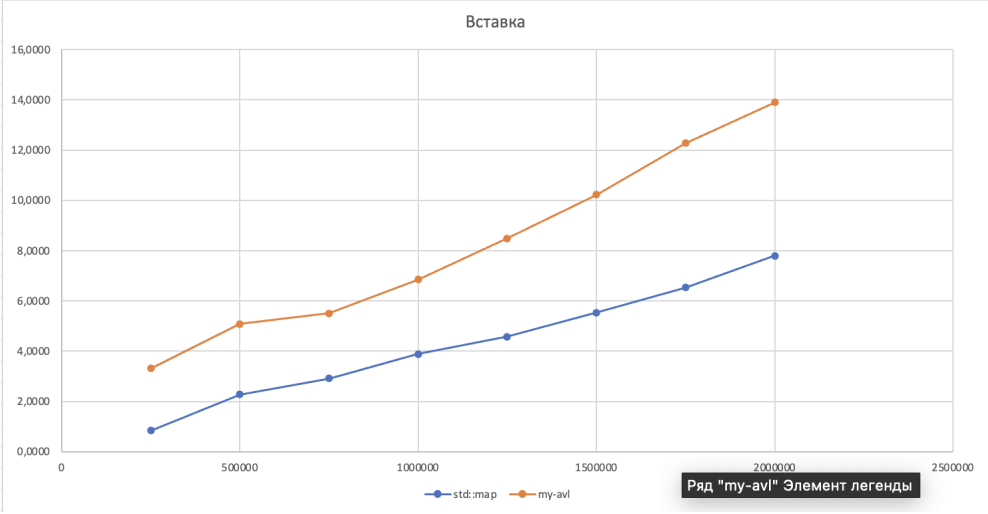

Рис. 1: Сравнение времени работы операции: Вставка

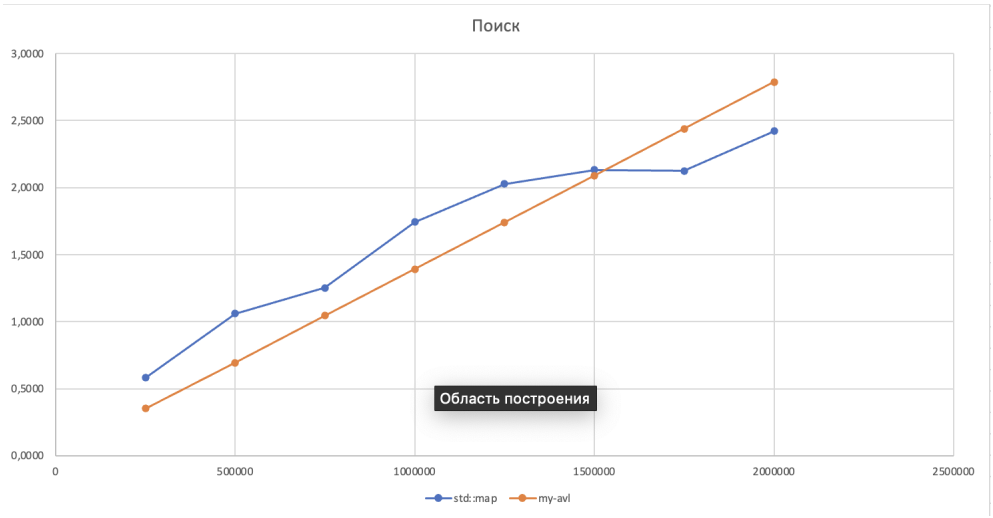

Рис. 2: Сравнение времени работы операции: Поиск

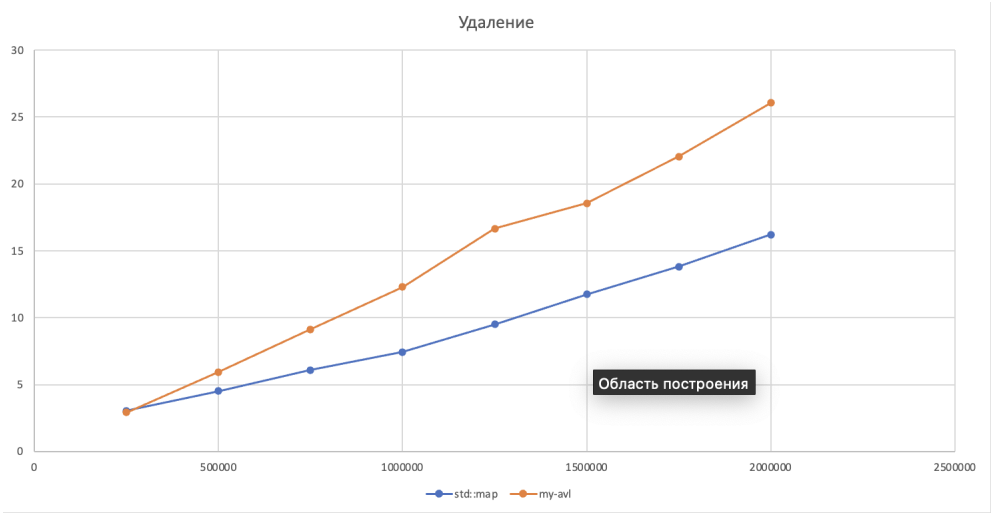

Рис. 3: Сравнение времени работы операции: Удаление

Выводы

Познакомился с сбалансированными леревьями. Данная структура данных может быть применена в качестве словаря, множества и подобных этим абстрактным типам данных. Так же в реализации простейших баз данных.