Лабораторная работа № 2 по курсу дискретного анализа: словарь

Выполнил студент группы М80-208Б-18 МАИ *Марков Александр.*

Условие

1. 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.

2. Вариант задания

   Вариант 1.

   Сортировка подсчётом.

   Тип дерева: AVL-дерево.

Метод решения

АВЛ-дерево — сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Данное дерево было решено реализовывать на основе двух структур TAvl и TAvlNode. В **структуре дерева** хранится только указатель на корень дерева.

Структура узла содержит:

- Ключ
- Значение
- Высоту узла
- Указатель на левого сына
- Указатель на правого сына

Балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев $|h(L) - h(R)| = 2$, изменяет связи предок-потомок в поддереве данной вершины так, чтобы восстановилось свойство дерева $|h(L) - h(R)| \leq 1$, иначе ничего не меняет. Есть **4 типа вращений** для балансировки:

1. Малое левое вращение.
2. Малое правое вращение.
3. Большое левое вращение.
4. Большое правое вращение.

Вставка элемента Пусть нам надо добавить ключ t. Будем спускаться по дереву, как при поиске ключа t. Если мы стоим в вершине a и нам надо идти в поддерево, которого нет, то делаем ключ t листом, а вершину a его корнем. Дальше поднимаемся вверх по пути поиска и пересчитываем баланс у вершин. Если мы поднялись в вершину i из левого поддерева, то $diff[i]$ увеличивается на единицу, если из правого, то уменьшается на единицу. Если пришли в вершину и её баланс стал равным 2 или -2, то делаем одно из четырёх вращений.

Так как в процессе добавления вершины мы рассматриваем не более, чем $O(h)$ вершин дерева, и для каждой запускаем балансировку не более одного раза, то суммарное количество операций при включении новой вершины в дерево составляет $O(logn)$ операций.

Удаление элемента Для простоты опишем рекурсивный алгоритм удаления. Если вершина — лист, то удалим её, иначе найдём самую близкую по значению вершину a, переместим её на место удаляемой вершины и удалим вершину a. От удалённой вершины будем подниматься вверх к корню и пересчитывать баланс у вершин. Если мы поднялись в вершину i из левого поддерева, то $diff[i]$ уменьшается на единицу, если из правого, то увеличивается на единицу.Если баланс стал равным 2 или -2, следует выполнить одно из четырёх вращений.

В результате указанных действий на удаление вершины и балансировку суммарно тратится, как и ранее, $O(h)$ операций. Требуемое количество действий — $O(logn)$.

Также был реализован тип TString для удобной работы с ключами.

Описание программы

Проект состоит из 4 файлов:

- main.cpp - главный файл, в котором реализована функция main
- avl.h - реализация AVL-дерева
- detail_avl.h - реализация операций с AVL-деревом в соответствии с заданием
- string.h - реализация строкового типа данных

Дневник отладки

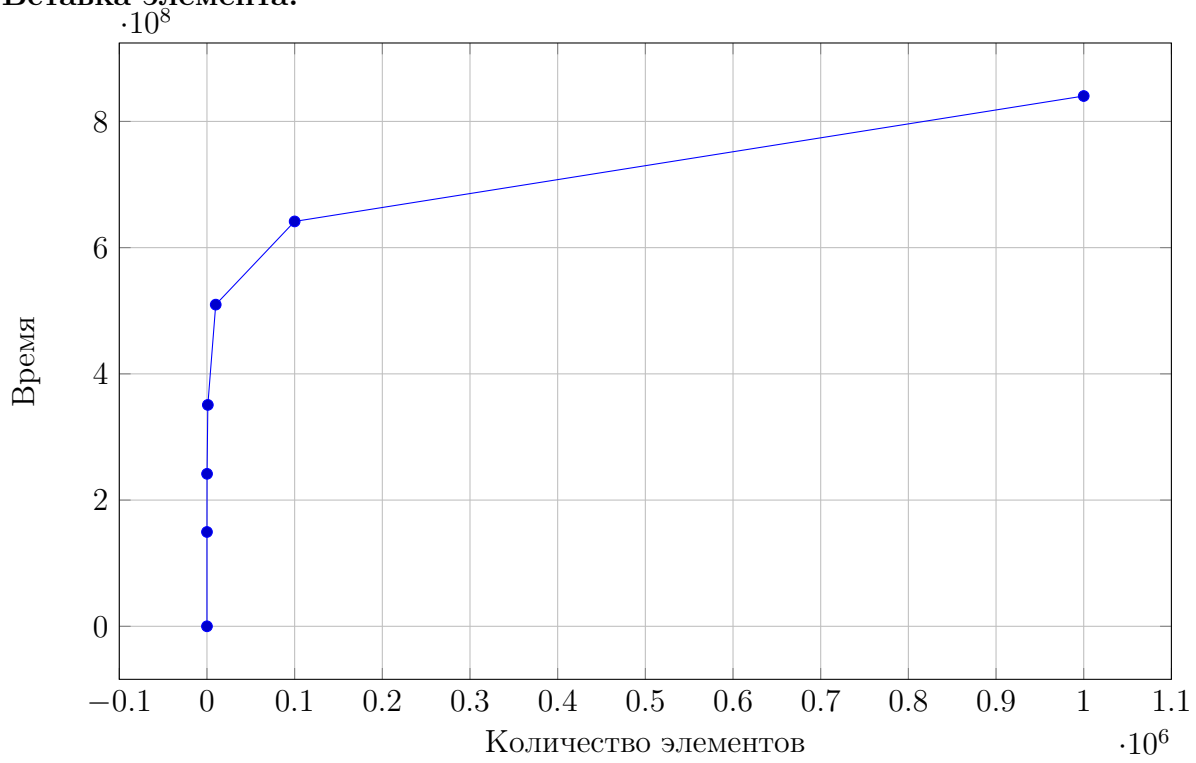
При создании этой таблицы была использована история посылок.

№	Время	Проблема	Описание
1-6	2019/11/15 - 2019/11/16	RE	Была утечка.
7-12	2019/11/17	RE	Неправильно работали функции вставки и удаления.
12-17	2019/11/17	TL	Неэффективно работала функция вставки. Реализовал строковый тип данных для удобства работы.

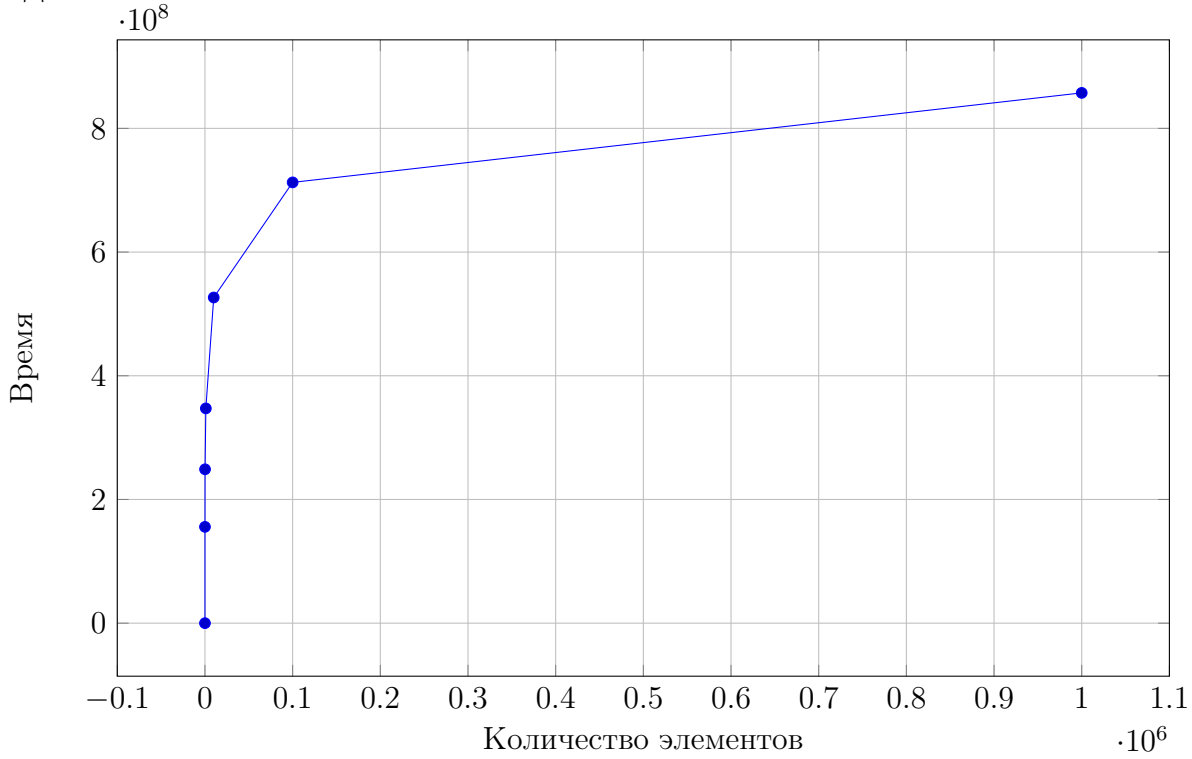
Тест производительности

Тесты создавались с помощью небольших программ benchmarkFind.cpp, benchmarkInsert.cpp, benchmarkRemove.cpp.

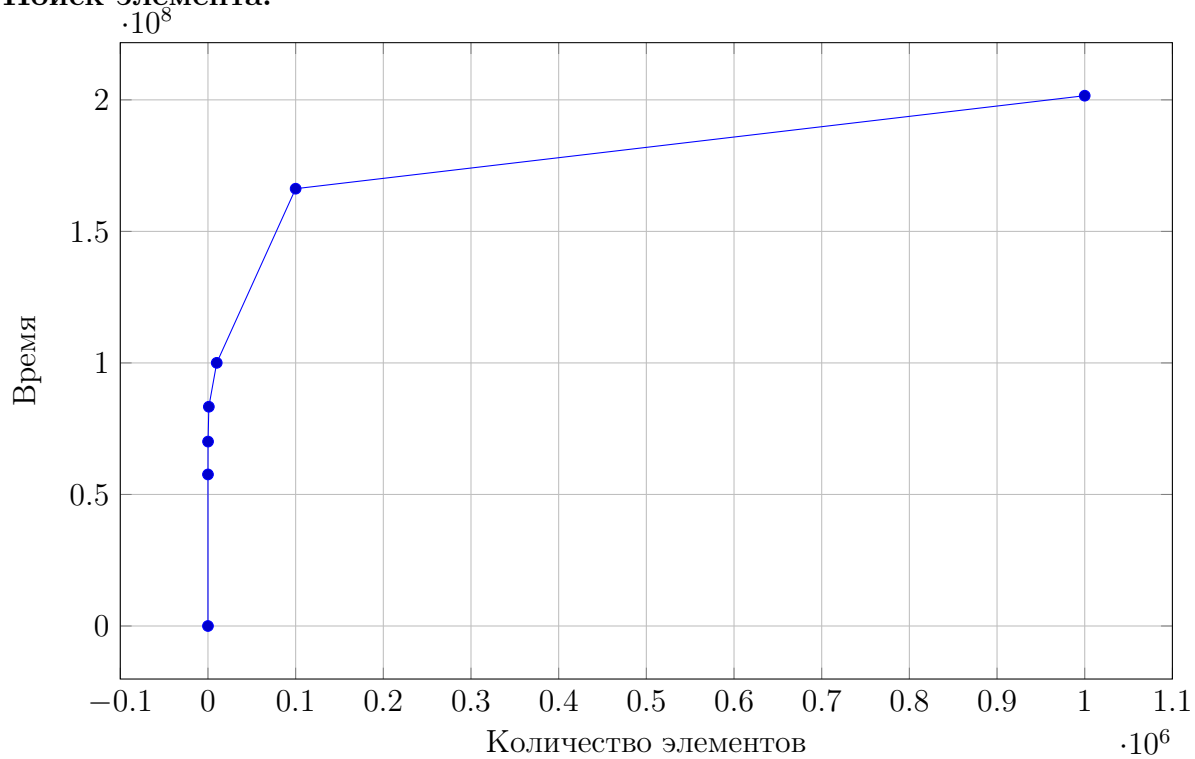
Вставка элемента:



Удаление элемента:



Поиск элемента:



Выводы

В данной лабораторной работе была реализована структура данных словарь, внутри которой скрыто AVL-дерево. Сложность вставки, поиска и удаления $O(log(n))$. Стоит использовать словарь, внутри которого AVL-дерево, если мы гарантированно хотим получить сложность $O(log(n))$ и данной скорости достаточно для поставленной задачи. На мой взгляд, структура данных AVL-дерево применяется редко, так как есть такая структура данных, как хэш-таблица, где те же операции работают в среднем за $O(1)$, хотя в худшем случпе может работать за $O(n)$, но это случается редко.