Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт № 8 информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Дискретный анализ»**

**Словарь**

Студент: Пермяков Никита Александрович

Группа: М80 – 208Б-19

Вариант: 1

Преподаватель: [Кухтичев Антон Алексеевич](https://mai.ru/education/schedule/ppc.php?guid=a3f854e1-f771-11e7-ae95-485b3919ee6d)

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

**Содержание**

1. Постановка задачи
2. Метод и алгоритм решения
3. Описание программы
4. Тесты производительности
5. Вывод

**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 264.

Тип дерева: AVL-дерево.

**Метод и алгоритм решения**

АВЛ-дерево — сбалансированное двоичное дерево поиска, в котором справедливо свойство:

* Для любой вершины высота двух поддеревьев различается не более чем на 1.

Данное дерево представлено на основе двух структур *TAvl* и *TAvlNode*.

В структуре дерева хранится только указатель на корень дерева.

Структура узла:

* 1. Ключ
  2. Значение
  3. Высоту узла
  4. Указатель на левого сына
  5. Указатель на правого сына

Балансировка вершины - операция, которая в случае разницы высоты левого и правого поддеревьев *abs(h(L) - h(R)) = 2*, изменяет связи предок-потомок в поддереве данной вершины так, чтобы восстановилось свойство дерева *abs(h(L) - h(R)) < 1*, иначе – связи не меняет.

Типы вращений для балансировки:

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

Вставка элемента

Добавим ключ *t*. Спускаемся по дереву в 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.

Далее поднимаемся вверх по пути поиска и пересчитываем баланс 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 делаем балансировку один раз, то суммарное количество операций при включении новой вершины в дерево будет *O(log n)*.

Удаление элемента (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.

Удаление вершины и балансировка - *O(h)* операций.

Требуемое количество действий *- O(log n).*

Также был реализован тип *TVector* для удобной работы с ключами.

**Описание программы**

Проект состоит из 4 файлов:

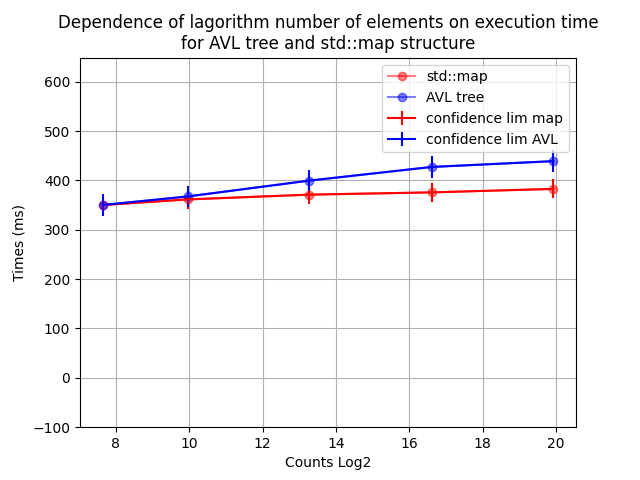
* main.cpp
* TAvl.h - само AVL-дерево
* ActionTAvl.h - операции с AVL-деревом
* TVector.h - реализация строкового типа данных.

Дневник отладки

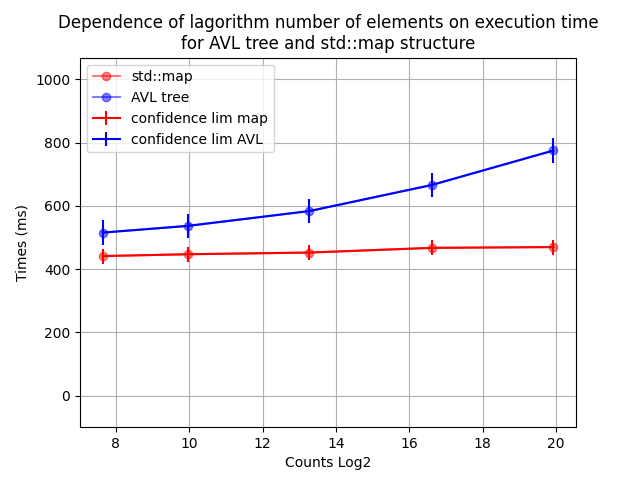
|  |  |  |  |
| --- | --- | --- | --- |
| № | Время | Проблема | Описание |
| 1 | 23:10 |  | Тест принимающей системы |
| 2-6 |  | Утечка ? | Неправильно работали функции вставки и удаления |
| 7 |  | Выход за границу (отрицательное число) | Исправлены типы данных |
| 8 |  |  | Написание тестов производительности |

**Тесты производительности**

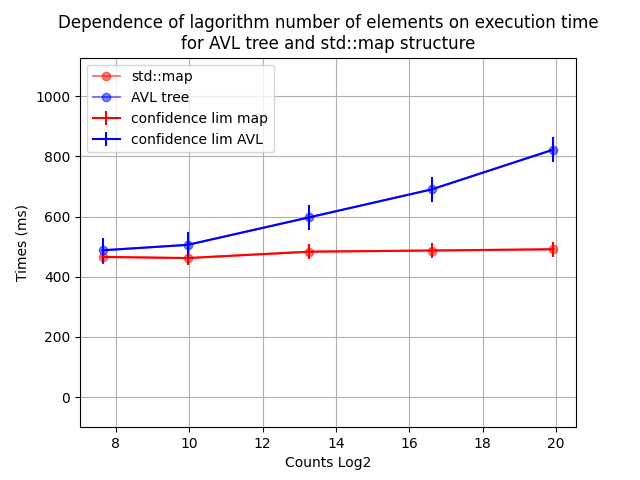
testFind.cpp – тестирование функции поиска



testInsert.cpp – тестирование функции вставки



testRemove.cpp – тестирование функции удаления

****

graph.py – скрипт формирования графиков

**Выводы**

В лабораторной работе была реализована структура данных AVL-дерево.

Сложность вставки, поиска и удаления O(log(n)).

AVL-дерево применяется реже, чем хэш-таблица, где те же операции работают за O(1). Они требуют дополнительных затрат на поддержание сбалансированности при вставке или удалении узлов. Если в дереве постоянно происходят вставки и удаления элементов, эти операции могут значительно снизить быстродействие. С другой стороны, если данные превращают бинарное дерево поиска в вырожденное, то теряется поисковая эффективность и тогда используется AVL-дерево.

Для AVL-дерева не существует наихудшего случая, так как оно является почти полным бинарным деревом.