

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА
по дисциплине «Алгоритмы и структуры данных»
Тема: Исследование операций вставки и исключения в AVL-деревьях

Студентка гр. 9304

Аксёнова Е.А.

Преподаватель

Филатов Ар.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студентка Аксёнова Е.А.

Группа 9304

Тема работы: Исследование операций вставки и исключения в AVL-деревьях

Исходные данные:

Содержание пояснительной записки:

- Аннотация
- Содержание
- Введение
- Формальная постановка задачи
- Описание структур данных и функций, описание алгоритма
- Тестирование
- Исследование
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 36 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 22.12.2020

Дата защиты реферата: 29.12.2020

Студентка		Аксёнова Е.А.
Преподаватель		Филатов Ар.Ю.

АННОТАЦИЯ

В данной курсовой работе производится исследование операций вставки и удаление элементов в структуре данных АВЛ-дерево. Исследование производится с помощью тестов для среднего и худшего случаев работы алгоритма. Результатами исследования являются числовые метрики, на основе которых формируются графики для сравнения с теоретическими оценками.

SUMMARY

The course work is brief investigation on insertion and deletion operations in data structure AVL-tree. All researches in work were conducted with test for worst and average cases. The results are numerical statistics compared with theoretical estimation.

СОДЕРЖАНИЕ

	Введение	5
1.	Формальная постановка задачи	6
2.	Ход выполнения работы	7
2.1.	Описание алгоритма	7
2.2.	Описание структур данных и функций	8
3.	Тестирование	11
4.	Исследование	16
4.1.	Исследование операций на АВЛ дереве	16
4.1.1	Вставка	16
4.1.2	Удаление	16
4.2	План исследования	17
4.3	Результаты исследования	18
	Заключение	21
	Список использованных источников	22
	Приложение А. Исходный код	23

ВВЕДЕНИЕ

Цель работы: Изучить структуру данных АВЛ дерево. Реализация и экспериментальное машинное исследование алгоритмов вставки и удаления в АВЛ дереве

Задача: Реализовать программу, которая считывает массив элементов АВЛ дерева и изменяет его в соответствии с выбором пользователя, управление осуществляется посредством командной строки.

1. ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Реализовать структуру данных АВЛ-дерево и провести исследование работы операций вставки и исключения (в среднем и худшем случае), чтобы проверить теоретическую оценку работы этих операций.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Описание алгоритма

Алгоритм добавления элемента в АВЛ-дерево:

Спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа.

Далее происходит вставка ключа. После вставки производится балансировка дерева.

Алгоритм удаления элемента из АВЛ-дерева:

Находим узел p с заданным ключом k (если не находим, то делать ничего не надо).

Далее алгоритм зависит от наличия правого поддерева у найденного узла. Если правого поддерева нет, то у узла может быть либо только один дочерний узел (дерево высоты 1), либо этот узел не имеет поддеревьев. В любом из этих случаев достаточно удалить найденный узел, а вместо него поместить указатель на его левое поддерево. Если же правое поддерево есть, то Находим минимальный ключ в этом поддереве. Опять же, по свойству АВЛ-дерева у минимального элемента справа либо подвешен единственный узел, либо там пусто. В обоих случаях надо просто вернуть указатель на правый узел и по пути назад (при возвращении из рекурсии) выполнить балансировку. Далее копируем правое и левое поддерево удаляемого узла в правое и левое поддерева минимального элемента соответственно и балансируем минимальное.

Алгоритм балансировки:

Вызываются повороты в зависимости от фактора баланса - разницей между высотой правого и левого поддерева. Если он равен -2, то проверяем левое поддерево. Если у левого поддерева проверяемого поддерева высота больше, чем у правого, то нам достаточно сделать один правый поворот, если же нет, то делаем последовательно сначала левый поворот для левого поддерева, и только потом правый поворот для всего. Если же фактор баланса

равен 2, то проверяем правое поддереву. Если у правого поддерева проверяемого поддерева высота больше, чем у левого, то нам достаточно сделать один левый поворот, если же нет, то делаем последовательно сначала правый поворот для правого поддерева, и только потом левый поворот для всего.

2.2. Описание структур данных и функций

1. Class Node - класс для представления узла дерева:

Поля:

T data – элемент, хранящийся в узле.

std::shared_ptr<Node<T>> left – указатель на левое поддерево

std::shared_ptr<Node<T>> right – указатель на правое поддерево

int height – высота узла

Методы:

T getData() - метод, который возвращает значение узла

2. Class Tree - класс для представления АВЛ-дерева:

Поля:

std::shared_ptr<Node<T>> root – указатель на корень дерева

int height – высота всего дерева

Методы:

std::shared_ptr<Node<T>> copyTree(std::shared_ptr<Node<T>>) – метод для копирования дерева по узлам

void fixhigh(std::shared_ptr<Node<T>>) – метод для поиска уровня каждого узла

bool findAndDelete(T) – метод, которая запускает поиск и удаление элемента

bool print() – метод, для вывода дерева

std::shared_ptr<Node<T>> createNode(std::vector<T>) – метод для создания дерева

bool findElem(std::shared_ptr<Node<T>>, T) – метод, который ищет элемент по значению

`std::shared_ptr<Node<T>> findMin(std::shared_ptr<Node<T>>)` – метод, который ищет минимальный элемент дерева

`std::shared_ptr<Node<T>> balanceRight(std::shared_ptr<Node<T>>, std::shared_ptr<Node<T>>)` – метод, который балансирует правое поддерево, если оно есть

`std::shared_ptr<Node<T>> deleteElem(std::shared_ptr<Node<T>>, T)` – метод, который удаляет элемент

`std::shared_ptr<Node<T>> rotateLeft(std::shared_ptr<Node<T>>)` – метод, реализующий левый поворот

`std::shared_ptr<Node<T>> rotateRight(std::shared_ptr<Node<T>>)` – метод, реализующий правый поворот

`std::shared_ptr<Node<T>> balanceTree(std::shared_ptr<Node<T>>)` – метод, который определяет нужно ли балансировать дерево и балансирует его

`void printElem (std::shared_ptr<Node<T>>, int)` – метод, который выводит дерево

3. Class Research - класс для проведения исследования:

Поля:

`std::vector<int> input` - вектор вставляемых и удаляемых элементов

Методы

`void generateAscendance()` - метод, генерирующий возрастающие значения

`void generateRandom(std::vector<int>& input, int lower, int upper)` - метод, генерирующий рандомные значения

`void runAdd(Tree<T> tree)` - метод, запускающий операцию вставки элементов для исследования

`void runDelete(Tree<T> tree)` - метод, запускающий операцию удаления для исследования

4. Функции:

`void checkStr(std::string&)` – функция для проверки введенной строки

Исходный код представлен в приложении А.

3. ТЕСТИРОВАНИЕ

Входные данные: на вход программе подается строка из элементов АВЛ-дерево, которые разделены пробелами. Затем программа принимает на вход число 1 или 2, который определяет дальнейшие действия программы (1 - вставка элемента, 2 - удаление элемента). А затем элемент, который требуется вставить или удалить.

Выходные данные: программа выводит АВЛ-дерево с удаленным или вставленным элементом. Либо выводит сообщение о том, что удаляемого элемента нет в дереве, и выводит исходное дерево.

Дерево выводится справа налево.

Тестирование производится с помощью скрипта на языке программирования Python. Исходный код представлен в приложении А.

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Результат проверки
1	1 2 3 4 5 1 2 3 1	<p>The AVL-Tree:</p> <pre> 1 2 3 4 5 </pre> <p>The AVL-Tree:</p> <pre> 1 1 2 3 4 5 </pre> <p>The AVL-Tree:</p> <pre> 1 1 2 2 3 4 5 </pre>	Finished right

		<p>The AVL-Tree:</p> <pre> 1 1 2 2 3 3 4 5 </pre>	
2	<p>7 5 9 1 5 3 7 5 1 9</p> <p>5 3 2 4 8 5</p> <p>5 4 5</p> <p>2</p>	<p>The AVL-Tree:</p> <pre> 1 1 2 3 4 5 5 5 7 7 8 9 9 </pre> <p>The AVL-Tree:</p> <pre> 1 1 2 3 4 5 5 5 7 7 8 9 9 </pre> <p>5</p> <p>The AVL-Tree:</p>	Finished right

		<pre> 1 1 2 3 3 5 5 5 7 7 8 9 9 </pre> <p>The AVL-Tree:</p> <pre> 1 1 2 3 3 5 5 7 8 9 9 </pre>	
3	3 4 5 6 6 3 2 4 3	<p>The AVL-Tree:</p> <pre> 2 3 3 3 4 4 5 6 6 </pre> <p>The AVL-Tree:</p> <pre> 2 3 3 3 4 </pre>	Finished right

		<pre> 4 5 6 6 7 </pre>	
4	<pre> 6 " 7 * 7 2 9 6 4 3 5 4 1 1 </pre>	<p>The AVL-Tree:</p> <pre> 2 3 4 4 5 6 7 7 9 </pre> <p>The AVL-Tree:</p> <pre> 1 2 3 4 4 5 6 7 7 9 </pre>	Finished right
5	<pre> 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 </pre>	<p>The AVL-Tree:</p> <pre> 1 1 1 1 1 1 1 1 1 1 </pre> <p>The AVL-Tree:</p> <pre> 1 1 </pre>	Finished right

		<pre> 1 / \ 1 1 / \ \ 1 1 1 / \ \ 1 1 1 </pre> <p>The AVL-Tree:</p> <pre> 1 / \ 1 1 / \ \ 1 1 1 / \ \ 1 1 1 </pre> <p>The AVL-Tree:</p> <pre> 1 / \ 1 1 / \ \ 1 1 1 / \ \ 1 1 1 </pre> <p>The AVL-Tree:</p> <pre> 1 / \ 1 1 / \ \ 1 1 1 / \ \ 1 1 1 </pre>	
--	--	--	--

4. ИССЛЕДОВАНИЕ

4.1. Исследование операций на АВЛ-дереве

4.1.1. Вставка

Пусть следует вставить ключ a . Будем спускаться по дереву, как при поиске ключа a . Если мы стоим в узле p , и над надо идти в поддереву, которого нет, то делает ключ a листом, а узел p его предком. Далее поднимаемся вверх по пути поиска и пересчитываем фактор баланса у узлов. Если поднялись в узел i из правого поддерева, то увеличиваем фактор баланса на единицу, если из левого, то уменьшаем. Если пришли в узел и фактор его баланса стал равным нулю, то значит, что высота поддерева не изменилась и можем прекратить подъём. Если пришли в узел, и его фактор баланса стал равным 1 или -1, то значит, что высота поддерева изменилась и мы продолжаем подъём. Если пришли в узел, и его фактор баланса стал равным 2 или -2, то делаем одно из 4 вращений, если после вращения баланс стал равен 0, то заканчиваем подъём, иначе продолжаем подъём.

Так в процессе вставки ключа мы рассматриваем не более, чем $O(h)$ узлов дерева, и для каждого узла запускаем балансировку не более одного раза, тогда суммарное количество операций при вставке элемента в АВЛ-дерево составляет $O(\log n)$ операций, где n - число элементов дерева.

4.1.2. Удаление

Если удаляемый узел a не имеет поддеревьев, то просто удалим его, иначе найдем самый близкий по значению к a узел b (самый левый узел правого поддерева), переместим узел b на место удаляемого узла a и удалим узел a . От удаленной вершины будем подниматься вверх к корню и пересчитывать значение фактора баланса у каждого узла. Если поднялись в узел i из правого поддерева, то уменьшаем фактор баланса на единицу, если из левого, то увеличиваем. Если пришли в узел и фактор его баланса стал равным нулю, то значит, что высота поддерева не изменилась и можем прекратить подъём. Если пришли в узел, и его фактор баланса стал равным 1 или -1, то значит, что высота поддерева

изменилась и мы продолжаем подъем. Если пришли в узел, и его фактор баланса стал равным 2 или -2, то делаем одно из 4 вращений, если после вращения баланс стал равен 0, то заканчиваем подъем, иначе продолжаем подъем

Так в процессе удаления элемента на удаление и балансировку суммарно тратится, как и в вставке, $O(h)$ операций, тогда суммарное количество операций при вставке элемента в АВЛ-дерево составляет $O(\log n)$ операций, где n - число элементов дерева.

На рисунке 1 показаны сложности операций для АВЛ-дерева.

	В среднем случае	В худшем случае
Расход памяти	$O(n)$	$O(n)$
Поиск	$O(\log n)$	$O(\log n)$
Вставка	$O(\log n)$	$O(\log n)$
Удаление	$O(\log n)$	$O(\log n)$

Рисунок 1 - Асимптотика работы операций в АВЛ-дереве

4.2. План исследования

Для подтверждения теоретической оценки был создан класс Research, который генерирует входные данные двух типов - строго возрастающей последовательности и случайной последовательности. Каждая из видов последовательности используется для операций вставки в АВЛ-дерево и удаления из него. Так в дерево вставляется набор элементов, а затем же он из него удаляется. Во время исследования работы операций вставки и удаления фиксируется количество вызовов соответствующих функций, в зависимости от высоты дерева.

4.3. Результаты исследования

На основе числовых метрик были построены графики. Ниже приведены 4 графика, иллюстрирующие асимптотику выполнения операций вставки и удаления в среднем и худшем случаях. На всех графиках оранжевой линией построен график логарифма от количества элементов в дереве (примерное значение высоты дерева). Синей кривой построена кривая, отображающая количество вызовов соответствующих функций. Этот график помогает наглядно показать схожесть формы графиков.

Наглядная визуализация поведения операций вставки и исключения продемонстрирована с помощью графиков, построенных программной, написанной на языке программирования Python. Исходный код представлен в приложении А. На рисунках 2 – 5 представлены графики вызовов функций вставки и исключения в зависимости от высоты дерева.

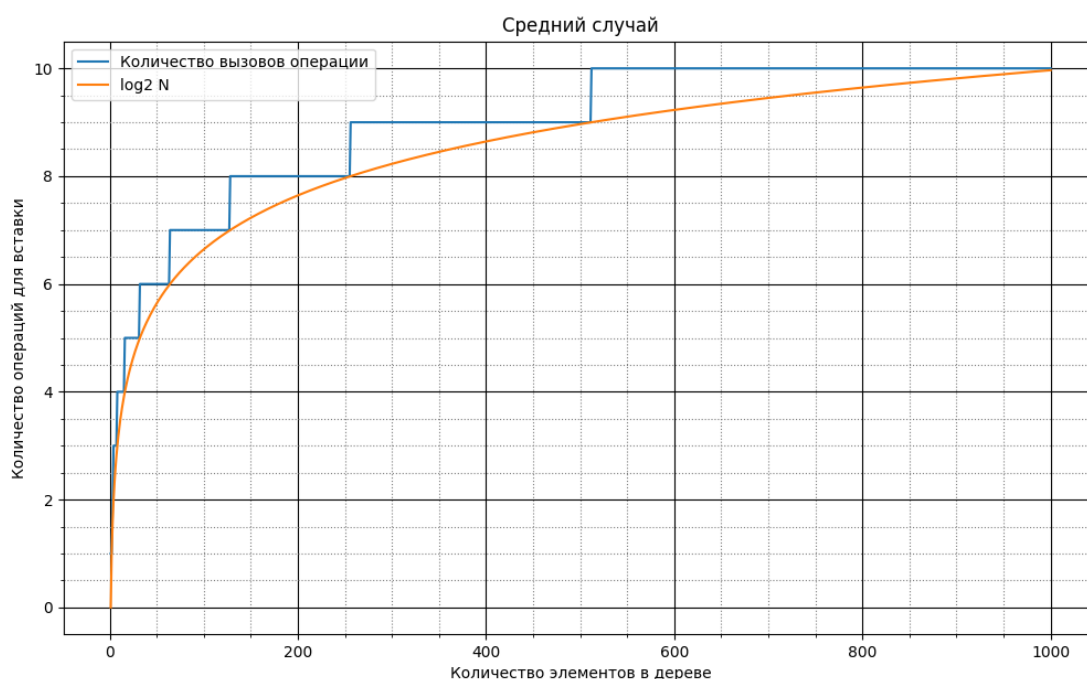


Рисунок 2 - Количество вызовов операций вставки в среднем случае в зависимости от высоты дерева

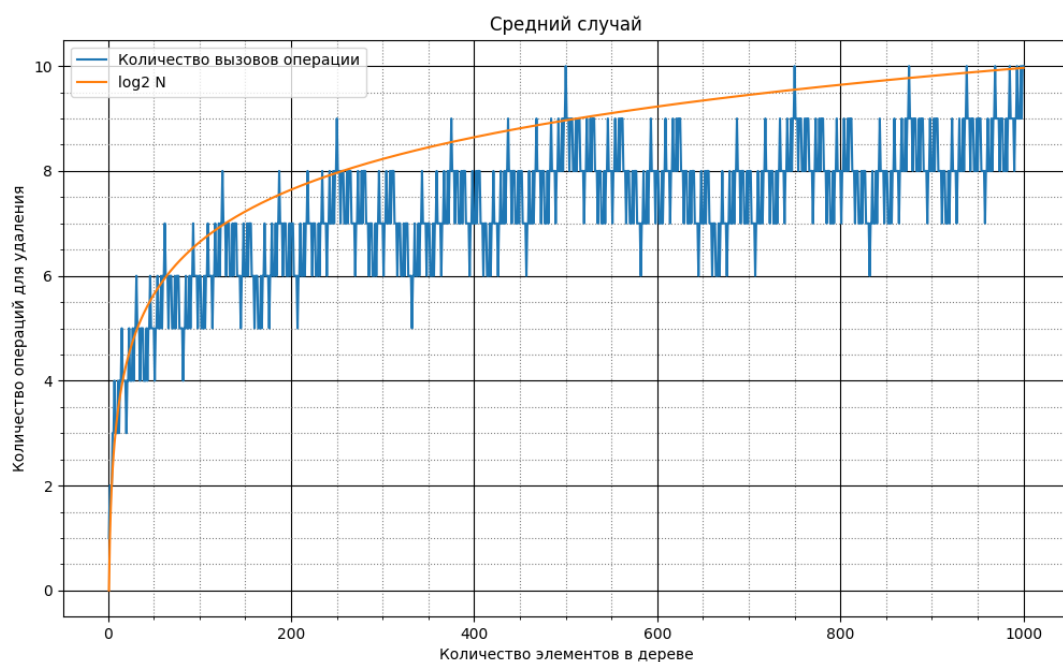


Рисунок 3 - Количество вызовов операций исключения в среднем случае в зависимости от высоты дерева

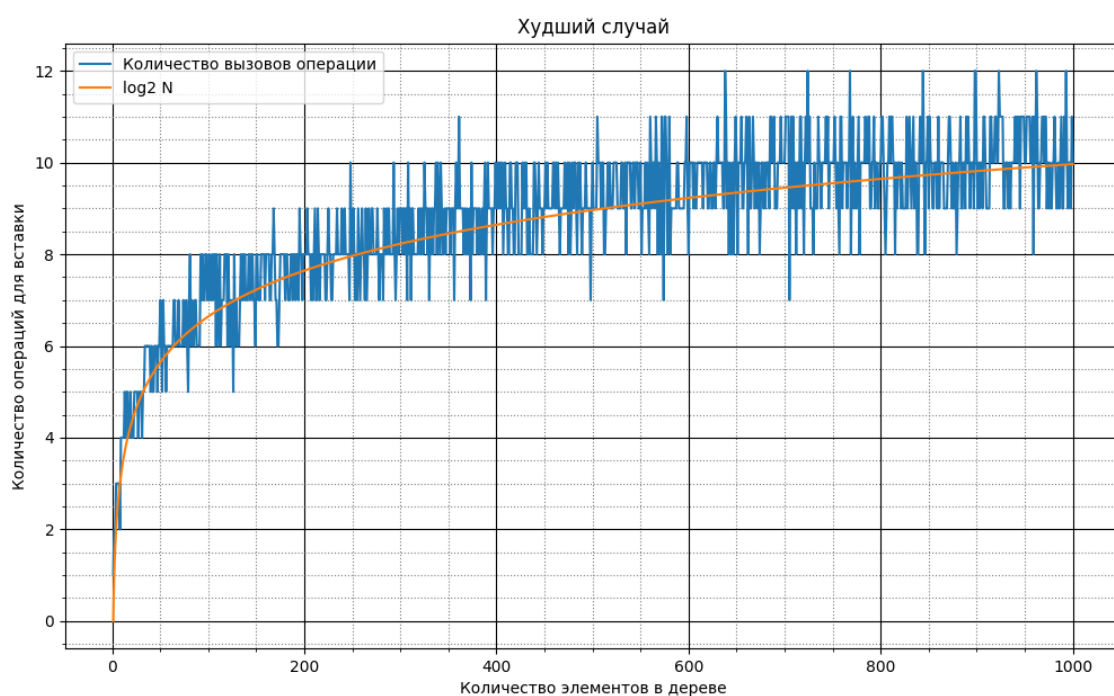


Рисунок 4 - Количество вызовов операций вставки в худшем случае в зависимости от высоты дерева

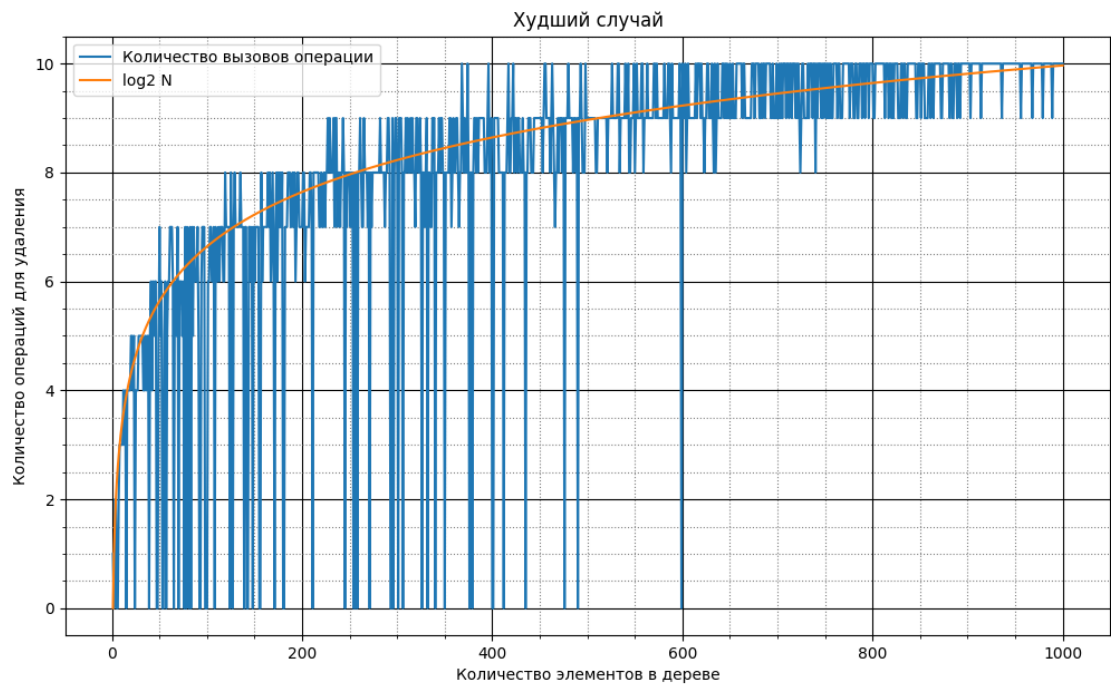


Рисунок 5 - Количество вызовов операций исключения в худшем случае в зависимости от высоты дерева

На графиках зависимости количества вызовов определенных функций от количества элементов, представленных выше, видно, что количество вызовов операций примерно равно $\log_2(n)$, где n - количество элементов в дереве, что подтверждает теоретическую оценку сложности операций вставки и удаления в AVL-дереве.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была реализована структура АВЛ-дерево, а также операции вставки и удаления для него. На основе числовых метрик были построены графики.

Полученные практические результаты сравнили с теоретическими оценками. Таким образом, была доказана теоретическая оценка асимптотики работы операций вставки и удаления в АВЛ-дерево.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сенюкова О. В. Сбалансированные деревья поиска: Учебно-методическое пособие. — М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова (лицензия ИД N 05899 от 24.09.2001 г.); МАКС Пресс, 2014. — 68 с.
2. АВЛ-деревья URL: <https://habr.com/ru/post/150732/> (дата обращения: 11.12.2020).
3. АВЛ-дерево URL: <https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE> (дата обращения: 11.12.2020).
4. АВЛ-дерево URL: <https://kvodo.ru/avl-tree.html> (дата обращения: 11.12.2020).
5. AVL-дерево URL: <http://algcourse.cs.msu.su/wp-content/uploads/2010/12/Lect17.pdf> (дата обращения: 11.12.2020).

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ПРИЛОЖЕНИЯ

Имя файла: main.cpp

```
#include <string>
#include <memory>
#include <sstream>
#include <iostream>
#include <cmath>
#include <fstream>
#include <algorithm>
#include <random>
#include <vector>
#include <iomanip>
#include <numeric>
#include <chrono>
```

```
#define inputSize 1000
static int operationCount = 0;
static int rotationCount = 0;
```

```
template<typename T>
class Tree;
```

```
template<typename T>
class Node {
public:
    Node(T data) : left(nullptr), right(nullptr), data(data) {
        height = 1;
    }

    Node(std::shared_ptr<Node<T>> left, std::shared_ptr<Node<T>> right, T
data) : left(left), right(right), data(data) {
        height = 1;
    }

    T getData(){
        if(this)
            return data;
    }
private:
    T data;
    std::shared_ptr<Node<T>> left, right;
    int height;

    friend class Tree<T>;
};
```

```
template<typename T>
class Tree {
public:
```

```

Tree(std::vector<T> vec) {
    std::sort(vec.begin(), vec.end());
    root = createNode(vec);
    height = root->height;
}

std::shared_ptr<Node<T>> copyTree(std::shared_ptr<Node<T>> tree) {
    if (tree->left != nullptr && tree->right != nullptr) {
        std::shared_ptr<Node<T>> node(new Node<T>(copyTree(tree-
>left), copyTree(tree->right), tree->data));
        return node;
    }
    if (tree->left != nullptr && tree->right == nullptr) {
        std::shared_ptr<Node<T>> node(new Node<T>(copyTree(tree-
>left), nullptr, tree->data));
        return node;
    }
    if (tree->left == nullptr && tree->right == nullptr) {
        std::shared_ptr<Node<T>> node(new Node<T>(nullptr, nullptr,
tree->data));
        return node;
    }
}

void fixhigh(std::shared_ptr<Node<T>> cur) {
    if (cur->left) {
        fixhigh(cur->left);
    }
    if (cur->right) {
        fixhigh(cur->right);
    }
    if (cur->left == nullptr) {
        cur->height = 1;
    }
    else {
        if (cur->right == nullptr) {
            cur->height = cur->left->height + 1;
        }
        else {
            cur->height = cur->left->height > cur->right->height ?
cur->left->height : cur->right->height;
            cur->height++;
        }
    }
}

Tree(const Tree<T>& tree) {
    std::cout << "I'm a copy constructor!\n";
    root = copyTree(tree.root);
    fixhigh(root);
    height = root->height;
}

```



```

}

Tree(Tree<T>&& tree) {
    std::cout << "I'm a move constructor!\n";
    std::swap(tree.root, root);
}

Tree<T>& operator = (const Tree<T>& tree) {
    root = copyTree(tree.root);
    return *this;
}

Tree<T>& operator = (Tree<T>&& tree) {
    root = std::move(tree.root);
    return *this;
}

std::shared_ptr<Node<T>> getRoot() {
    return this->root;
}

bool findAndDelete(T e) {
    operationCount++;
    if (findElem(root, e)) {
        root = deleteElem(root, e);
        if (root == nullptr) {
            height = 0;
        }
        else {
            height = root->height;
        }
        return true;
    }
    else {
        return false;
    }
}

bool print() {
    if (!this->height) {
        std::cout << "The tree is empty\n";
        return 0;
    }
    std::cout << "The AVL-Tree:" << '\n';
    printElem(root, height);
    return 1;
}

void insert(T e) {
    operationCount++;
    root = insertElem(root, e);
}

```

```

        fixhigh(root);
        height = root->height;
        root = balanceTree(root);
    }

private:

    std::shared_ptr<Node<T>> insertElem(std::shared_ptr<Node<T>> cur, T
e) {
operationCount++;
        if (!cur) {
            return std::make_shared<Node<T>>(e);
        }
        if (e <= cur->data) {
            cur->left = insertElem(cur->left, e);
        }
        else {
            cur->right = insertElem(cur->right, e);
        }

        return balanceTree(cur);
    }

    std::shared_ptr<Node<T>> createNode(std::vector<T> vec) {
        if (vec.size() == 0) {
            return nullptr;
        }
        else {
            int ind = vec.size() / 2;
            auto node = std::make_shared<Node<T>>(vec[ind]);
            std::vector<T> left, right;
            for (int i = 0; i < vec.size(); i++) {
                if (i < ind) {
                    left.push_back(vec[i]);
                }
                if (i > ind) {
                    right.push_back(vec[i]);
                }
            }
            node->left = createNode(left);
            node->right = createNode(right);
            if (node->left == nullptr) {
                node->height = 1;
            }
            else {
                if (node->right == nullptr) {
                    node->height = node->left->height + 1;
                }
                else {
                    node->height = node->left->height > node->right-
>height ? node->left->height : node->right->height;

```

```

        node->height++;
    }
}
return node;
}
}

bool findElem(std::shared_ptr<Node<T>> cur, T e) {
    if (cur == nullptr) {
        std::cout<<"Fuck\n";
        return false;
    }
    else if (cur->data > e) {
        return findElem(cur->left, e);
    }
    else if (cur->data < e) {
        return findElem(cur->right, e);
    }
    else {
        return true;
    }
}

std::shared_ptr<Node<T>> findMin(std::shared_ptr<Node<T>> cur) {
    operationCount++;
    if (cur == nullptr) {
        return nullptr;
    }
    else if (cur->left == nullptr) {
        return cur;
    }
    else {
        return findMin(cur->left);
    }
}

std::shared_ptr<Node<T>> balanceRight(std::shared_ptr<Node<T>> cur,
std::shared_ptr<Node<T>> min) {
    if (cur == nullptr) {
        return nullptr;
    }
    else if (cur->left == nullptr) {
        return cur->right;
    }
    else if (cur->left == min) {
        cur->left = nullptr;
        return balanceTree(cur);
    }
    else {
        cur->left = balanceRight(cur->left, min);
        return balanceTree(cur);
    }
}

```

```

    }
}

std::shared_ptr<Node<T>> deleteElem(std::shared_ptr<Node<T>> cur, T
e) {
    operationCount++;
    if (cur->data > e) {
        cur->left = deleteElem(cur->left, e);
    }
    else if (cur->data < e) {
        cur->right = deleteElem(cur->right, e);
    }
    else {
        if (cur->right == nullptr) {
            return cur->left;
        }
        else {
            auto min = findMin(cur->right);
            min->right = balanceRight(cur->right, min);
            min->left = cur->left;
            return balanceTree(min);
        }
    }
    return balanceTree(cur);
}

std::shared_ptr<Node<T>> rotateLeft(std::shared_ptr<Node<T>> cur) {
    auto right = cur->right;
    cur->right = right->left;
    right->left = cur;
    int lHeight = 0, rHeight = 0;
    if (cur->left != nullptr) {
        lHeight = cur->left->height;
    }
    if (cur->right != nullptr) {
        rHeight = cur->right->height;
    }
    cur->height = lHeight > rHeight ? lHeight + 1 : rHeight + 1;
    lHeight = rHeight = 0;
    if (right->left != nullptr) {
        lHeight = right->left->height;
    }
    if (right->right != nullptr) {
        rHeight = right->right->height;
    }
    right->height = lHeight > rHeight ? lHeight + 1 : rHeight + 1;
    return right;
}

std::shared_ptr<Node<T>> rotateRight(std::shared_ptr<Node<T>> cur) {
    auto left = cur->left;

```

```

    cur->left = left->right;
    left->right = cur;
    int lHeight = 0, rHeight = 0;
    if (cur->left != nullptr) {
        lHeight = cur->left->height;
    }
    if (cur->right != nullptr) {
        rHeight = cur->right->height;
    }
    cur->height = lHeight > rHeight ? lHeight + 1 : rHeight + 1;
    lHeight = rHeight = 0;
    if (left->left != nullptr) {
        lHeight = left->left->height;
    }
    if (left->right != nullptr) {
        rHeight = left->right->height;
    }
    left->height = lHeight > rHeight ? lHeight + 1 : rHeight + 1;
    return left;
}

std::shared_ptr<Node<T>> balanceTree(std::shared_ptr<Node<T>> cur) {
    rotationCount++;
    int lHeight = 0, rHeight = 0;
    if (cur->left != nullptr) {
        lHeight = cur->left->height;
    }
    if (cur->right != nullptr) {
        rHeight = cur->right->height;
    }
    cur->height = lHeight > rHeight ? lHeight + 1 : rHeight + 1;
    int diff = rHeight - lHeight;
    if (diff == 2) {
        int diffRight = 0;
        if (cur->right->left != nullptr) {
            diffRight -= cur->right->left->height;
        }
        if (cur->right->right != nullptr) {
            diffRight += cur->right->right->height;
        }
        if (diffRight < 0) {
            cur->right = rotateRight(cur->right);
        }
        return rotateLeft(cur);
    }
    else if (diff == -2) {
        int diffLeft = 0;
        if (cur->left->left != nullptr) {
            diffLeft -= cur->left->left->height;
        }
        if (cur->left->right != nullptr) {

```

```

        diffLeft += cur->left->right->height;
    }
    if (diffLeft > 0) {
        cur->left = rotateLeft(cur->left);
    }
    return rotateRight(cur);
}
else {
    return cur;
}
}

void printElem(std::shared_ptr<Node<T>> cur, int level) {
    if (cur->left != nullptr) {
        printElem(cur->left, level - 1);
    }
    for (int i = 0; i < level; i++) {
        std::cout << '\t';
    }
    std::cout << cur->data << '\n';
    if (cur->right != nullptr) {
        printElem(cur->right, level - 1);
    }
}

std::shared_ptr<Node<T>> root;
int height;

};

void checkStr(std::string& str) {
    for (int i = 0; i < str.size(); i++) {
        if (!isdigit(str[i]) && str[i] != ' ') {
            str.erase(i, 1);
            i -= 1;
        }
    }
}

class Research {
public:
    std::vector<int> input;

    void generateAscendance();

    void generateRandom(std::vector<int>& input, int lower, int upper);

    template<typename T>
    void runAdd(Tree<T> tree);

    template<typename T>

```

```

        void runDelete(Tree<T> tree);

};

void Research::generateAscendance() {
    for (int i = 1; i <= inputSize; i++) {
        input.push_back(i);
    }
}

void Research::generateRandom(std::vector<int>& input, int lower, int
upper) {
    auto now = std::chrono::high_resolution_clock::now();
    std::mt19937 gen;
    gen.seed(now.time_since_epoch().count());
    std::uniform_int_distribution<> distribution(lower, upper);
    while (input.size() < inputSize) {
        input.push_back(distribution(gen));
    }
}

template<typename T>
void Research::runAdd(Tree<T> tree) {
    int treeSize = 0;
    std::ofstream outAdd, outRot;
    outAdd.open("resAdd.txt");
    outRot.open("resRotate.txt");
    std::vector<int> indices = this->input;
    for (auto x : indices) {
        treeSize++;
        operationCount = 0;
        rotationCount = 0;
        tree.insert(x);
        outAdd << treeSize << ' ' << operationCount-2 << "\n";
        outRot << treeSize << ' ' << rotationCount-1 << "\n";
    }
    outAdd.close();
    outRot.close();
}

template<typename T>
void Research::runDelete(Tree<T> tree) {
    std::ofstream out;
    int treeSize = inputSize;
    out.open("resDelete.txt");
    std::vector<int> indices = this->input;
    for (auto x : indices) {
        operationCount = 0;
        tree.findAndDelete(x);
        out << treeSize << ' ' << operationCount-1 << "\n";
        treeSize--;
    }
}

```

```

    }

    out.close();

}

typedef int elem;

int main(int argc, char* argv[]) {
    std::vector<elem> vec;
    std::string str;
    if (argc == 1) {
        std::getline(std::cin, str);
    }
    else if (argc >= 3) {
        str = argv[1];
    }
    else {
        std::cout << "Wrong input";
        std::cout << "Finished right\n";
        return 0;
    }
    checkStr(str);
    std::stringstream ss(str);
    elem value;
    while (ss >> value) {
        vec.push_back(value);

        if (ss.peek() == ' ') {
            ss.ignore();
        }
    }
    if (!vec.size()) {
        std::cout << "The tree is empty\n";
        std::cout << "Finished right\n";
        return 0;
    }
    Tree<elem> tree(vec);
    //Tree<elem> tree1(tree);
    if (!tree.print()) {
        std::cout << "Finished right\n";
        return 0;
    }
    //std::cout <<"This is copied tree:\n";
    /*if (!tree1.print()) {
        return 0;
    }*/
    std::string toInsert;
    std::string toDelete;
    int DoI;
    while (true) {

```



```

std::cout << '\n';
if (argc == 1) {
    std::cout << "Chose what you want to do: 1. Insert, 2.
Delete\n";
    std::cin >> DoI;
    if (DoI == 2) {
        std::cout << "Input element, that you want to delete:\n";
        std::cin >> toDelete;
    }
    if (DoI == 1) {
        std::cout << "Input element, that you want to insert:\n";
        std::cin >> toInsert;
    }
}
else {
    DoI = std::stoi(argv[3]);
    if (DoI == 2) {
        toDelete = argv[2];
    }
    if (DoI == 1) {
        toInsert = argv[2];
    }
}
if (DoI != 1 && DoI != 2) {
    break;
}
if (toDelete == "exit") {
    break;
}
if (toInsert == "exit") {
    break;
}
if (DoI == 2) {
    checkStr(toDelete);
}
if (DoI == 1) {
    checkStr(toInsert);
}

std::vector<elem> del;
std::stringstream ss;
if (DoI == 2) {
    ss << toDelete;
}
if (DoI == 1) {
    ss << toInsert;
}
//elem value;
while (ss >> value) {
    del.push_back(value);
}

```

```

        if (ss.peek() == ' ') {
            ss.ignore();
        }
    }
    while (del.size()) {
        elem num = del[0];
        del.erase(del.begin());
        std::cout << '\n';
        if(DoI == 2){
            if (tree.findAndDelete(num)) {
                if (!tree.print()) {
                    std::cout << "Finished right\n";
                    return 0;
                }
            }
            else {
                std::cout << num << " is not in tree\n";
                if (!tree.print()) {
                    std::cout << "Finished right\n";
                    return 0;
                }
            }
        }
        if(DoI == 1){
            tree.insert(num);
            if (!tree.print()) {
                std::cout << "Finished right\n";
                return 0;
            }
        }

        if (!del.size() && argc >= 3) {
            std::cout << "Finished right\n";
            return 0;
        }
    }
}

/*std::vector<elem> vec;
vec.push_back(1);
Tree<elem> tree(vec);
Research res;
//res.generateRandom(res.input, 0, inputSize);
res.generateAscendance();
Tree<elem> tree1(res.input);
res.runDelete(tree1);*/
std::cout << "Finished right\n";
return 0;
}

```

```

Имя файла: plot.py
import sys
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import math
from numpy import *

file = sys.argv[1]

fs = file + ".txt"

f = open(fs, 'r')

x, y = [0], [0]
l = 0

for l in f:
    row = l.split()
    tempx = int(row[0])
    tempy = int(row[1])
    x.append(tempx)
    y.append(tempy)

f.close()

fig, ax1 = plt.subplots(
    nrows=1, ncols=1,
    figsize=(12, 12)
)

t = linspace(0, max(x), max(x))
a = log(t)/log(2)

x.pop(0)
y.pop(0)
ax1.plot(x, y, label = 'Количество вызовов операции')
ax1.plot(t, a, label = 'log2 N')

ax1.grid(which='major',
        color = 'k')

ax1.minorticks_on()

ax1.grid(which='minor',
        color = 'gray',
        linestyle = ':')

ax1.legend()

```

```

ax1.set_xlabel('Количество элементов в дереве')

if(file == "resAdd"):
    ax1.set_ylabel('Количество операций для вставки')
else:
    ax1.set_ylabel('Количество операций для удаления')
fig.set_figwidth(15)
fig.set_figheight(10)

plt.title("Средний случай")
plt.show()

```

Имя файла: test.py

```

import unittest
import subprocess

```

```

class tester(unittest.TestCase):

    def test1(self):
        with open('./Tests/tests/test1.txt', 'r') as file:
            line = file.readlines()
            print("\tTest 1\n")
            print("Input:\n")
            print(line[0])
            print(line[1])
            print(line[2])
            self.assertIn("Finished right", subprocess.check_output(["./lab5",
                line[0], line[1], line[2]], universal_newlines=True))
            print("\nOutput:\n")
            print(subprocess.check_output(["./lab5", line[0], line[1], line[2]],
                universal_newlines=True))

    def test2(self):
        with open('./Tests/tests/test2.txt', 'r') as file:
            line = file.readlines()
            print("\tTest 2\n")
            print("Input:\n")
            print(line[0])
            print(line[1])
            print(line[2])
            self.assertIn("Finished right", subprocess.check_output(["./lab5",
                line[0], line[1], line[2]], universal_newlines=True))
            print("\nOutput:\n")
            print(subprocess.check_output(["./lab5", line[0], line[1], line[2]],
                universal_newlines=True))

    def test3(self):
        with open('./Tests/tests/test3.txt', 'r') as file:
            print("\tTest 3\n")
            line = file.readlines()

```

```

print("Input:\n")
print(line[0])
print(line[1])
print(line[2])
self.assertIn("Finished right", subprocess.check_output(["./lab5",
line[0], line[1], line[2]], universal_newlines=True))
print("\nOutput:\n")
print(subprocess.check_output(["./lab5", line[0], line[1], line[2]],
universal_newlines=True))
def test4(self):
with open('./Tests/tests/test4.txt', 'r') as file:
print("\tTest 4\n")
line = file.readlines()
print("Input:\n")
print(line[0])
print(line[1])
print(line[2])
self.assertIn("Finished right", subprocess.check_output(["./lab5",
line[0], line[1], line[2]], universal_newlines=True))
print("\nOutput:\n")
print(subprocess.check_output(["./lab5", line[0], line[1], line[2]],
universal_newlines=True))
def test5(self):
with open('./Tests/tests/test5.txt', 'r') as file:
line = file.readlines()
print("\tTest 5\n")
print("Input:\n")
print(line[0])
print(line[1])
print(line[2])
self.assertIn("Finished right", subprocess.check_output(["./lab5",
line[0], line[1], line[2]], universal_newlines=True))
print("\nOutput:\n")
print(subprocess.check_output(["./lab5", line[0], line[1], line[2]],
universal_newlines=True))
def test6(self):
with open('./Tests/tests/test6.txt', 'r') as file:
line = file.readlines()
print("\tTest 6\n")
print("Input:\n")
print(line[0])
print(line[1])
print(line[2])
self.assertIn("Finished right", subprocess.check_output(["./lab5",
line[0], line[1], line[2]], universal_newlines=True))
print("\nOutput:\n")
print(subprocess.check_output(["./lab5", line[0], line[1], line[2]],
universal_newlines=True))
if __name__ == '__main__':

```