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 АВЛ-деревьях

Студентка гр. 9382 _____ Балаева М.О

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Балаева М.О.

Группа 9382

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

Исходные данные:

АВЛ-деревья - вставка и исключение. Исследование (в среднем, в худшем случае(Последовательная вставка от 1 до 10000))

Содержание пояснительной записки:

титульный лист, лист задания, аннотация, содержание, формальная постановка задачи, описание алгоритма, описание структур данных и функций, тестирование, исследование, программный код (в приложении) с комментариями, выводы.

Предполагаемый объем пояснительной записки:

Не менее 12 страниц.

Дата выдачи задания: 01.09.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 29.12.2020

Студентка _____ Балаева М.О.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

В данной курсовой работе производилось исследование структуры данных “АВЛ-дерево”, а также операция по вставке и удалению элементов из нее. Исследование проходило с помощью тестов для разных случаев поведения алгоритмов, в среднем, худшем случае. Результатом исследования являются числовые метрики, на основе которых формируется статистика для сравнения с теоретическими оценками.

SUMMARY

This paper is supposed to bring brief investigation on abstract data structure AVL-tree and its insertion and deletion operations. Research provided with tests on different cases of structure behavior, e.g. worst, average. The result is numerical statistics compared with theoretical estimation.

Содержание

ВВЕДЕНИЕ

Двоичное дерево поиска — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

● Оба поддерева — левое и правое — являются двоичными деревьями поиска.

● У всех узлов левого поддерева произвольного узла X значения ключей данных меньше либо равны, нежели значение ключа данных самого узла X.

● У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше.

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

АВЛ — аббревиатура, образованная первыми буквами фамилий создателей (советских учёных) Георгия Максимовича Адельсон-Вельского и Евгения Михайловича Ландиса.

Цель работы

В данной курсовой работе основной целью является исследование операций вставки и исключения в АВЛ-деревьях.

Для реализации данной цели необходимо решить следующие задачи:

1. Собрать теоретические сведения по изучаемой структуре данных и функциях для работы с ней.
2. Написать программу , реализующую заданную структуру данных и необходимые алгоритмы, а также провести исследование.

1. ЗАДАНИЕ

Вариант 16. Реализовать структуру данных “АВЛ-дерево” и провести исследование работы операций вставки и исключения(в среднем, в худшем случае) дабы подтвердить теоретическую оценку работы этих операций.

2. ОПИСАНИЕ АЛГОРИТМОВ

Алгоритм добавления элемента следующий:

1. Вставка элемента происходит почти также, как и обычном БДП. Спускаемся по дереву вниз, сравнивая элемент для вставки с элементами дерева.
2. После вставки необходимо сбалансировать дерево.

Балансировка дерева происходит, когда разница между высотами поддеревьев одного элемента становится равной 2. В таком случае, в зависимости от конфигурации, необходимо провести серию вращений.

Вставка нового ключа в АВЛ-дерево выполняется, по большому счету, так же, как это делается в простых деревьях поиска: спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. Единственное отличие заключается в том, что при возвращении из рекурсии (т.е. после того, как ключ вставлен либо в правое, либо в левое поддерево, и это дерево сбалансировано) выполняется балансировка текущего узла.

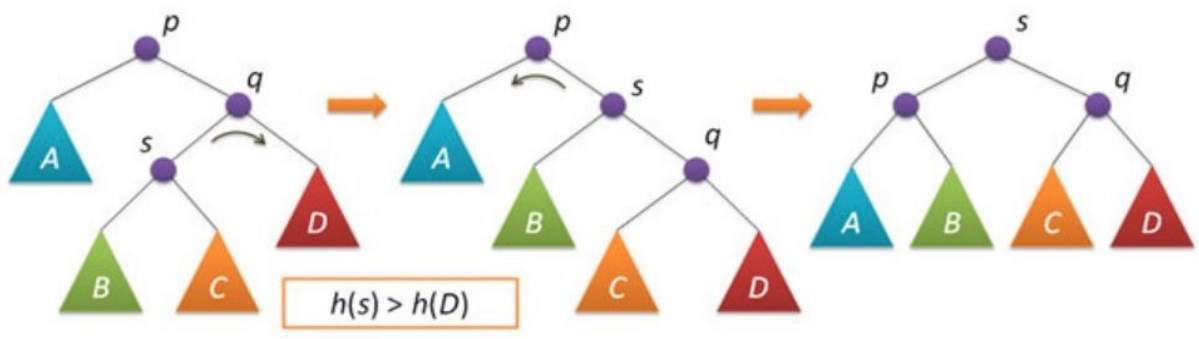

Рисунок 1. - Балансировка дерева с помощью правого и левого вращения.

Алгоритм удаления элемента **:**

Находим узел p с заданным ключом k (если не находим, то делать ничего не надо), в правом поддереве находим узел min с наименьшим ключом и заменяем удаляемый узел p на найденный узел min. При каждом выходе из рекурсии необходимо ребалансировать дерево.

Идея следующая: находим узел p с заданным ключом k (если не находим, то делать ничего не надо), в правом поддереве находим узел min с наименьшим ключом и заменяем удаляемый узел p на найденный узел min.

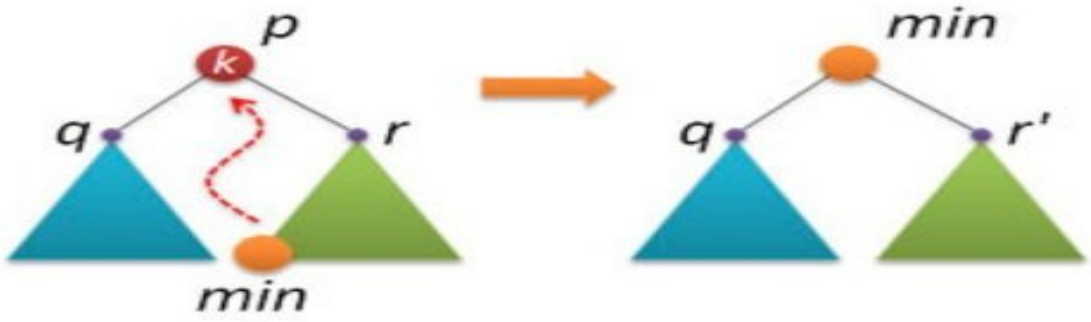

Рисунок 2- Алгоритм удаления элемента

3.ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ.

3.1. Класс АВЛ-дерева

class AVL_tree - Класс представления АВЛ-дерева. В классе AVL_tree определены функции вставки и исключения. Поля класса : Node *root - корень дерева. Является классом-оберткой над Node , в то время когда в Node определены функции — вращения, балансировки. Поля класса : int value - значение в узле, int height - уровень узла, Node *lt - указатель на левого сына, Node *rt - указатель на правого сына Node *rotate_right() - правое вращение, Node *rotate_left() - левое вращение.

3.2. Описание функций

int diff_height() - Функция поиска разности между высотами поддеревьем элемента. Возвращает между высотами левого и правого поддерева.

void update_height() - После каждой вставки/балансировки/удаления нужно обновлять высоту дерева.

Node *rotate_right() - Правое вращение. Возвращает Node* p - новый корень полученного дерева.

Node *rotate_left() - Функция левого вращения. Возвращает новый корень дерева.

Node *balance() - Функция балансировки АВЛ-дерева. Балансировка нужна в случае когда разница высот левого и правого поддеревьев становится равной |2|. Возвращает указатель на самого себя(узел).

AVL_tree - Конструктор АВЛ-дерева принимает корень.

void print_tree(Node *node, int level) - Служебная функция вывода дерева. Выводит дерево не сверху-вниз, а слева-направо. Принимает корень выводимого поддерева. Принимает уровень рекурсии для индентации.

Node *insert_node(Node *node, int value) - Вставка элемента. В конце необходимо балансировать. Принимает корень дерева, куда добавляем. Принимает ключ элемента. Возвращает корень сбалансированного дерева.

Node *remove_node(Node *node, int value) - Функция удаления элемента с заданным ключом находим узел p с заданным ключом value , в правом поддереве находим узел min с наименьшим ключом и заменяем удаляемый узел p на найденный узел min. Принимает корень дерева, в котором происходит удаление элемента. Принимает value ключ для удаления. Возвращает ребалансированный корень дерева.

Node *find_min(Node *node) - Функция поиска минимального элемента в дереве или поддереве. Возвращает корень дерева, где ищется минимум. Возвращает указатель на элемент с наименьшим ключем.

Node *remove_min(Node *node) - Удаление минимального элемента из заданного дерева. По свойству АВЛ- дерева у минимального элемента справа либо подвешен узел, либо там пусто. В обоих случаях надо просто вернуть

указатель на правый узел и при возвращении из рекурсии выполнить балансировку. Принимает корень дерева или поддерева, где удаляется минимальный элемент. Возвращает указатель на новый корень после балансировки.

Node *lets_insert_node(Node *root, int value) - Служебная функция-обертка над вставком для удобного вывода. Возвращает корень дерева или поддерева, куда вставляется элемент. Принимает ключ элемента для вставки. Возвращает корень поддерева.

Node *lets_remove_node(Node *root, int value) - Служебная функция-обертка над remove.

Принимает поддерево или дерево, в котором удаляется элемент ,элемент для удаления. Возвращает корень дерева, где удаляли элемент.

4. ТЕСТИРОВАНИЕ

Основной тест No1:

Входные данные: Создать дерево с корнем 35. Вставить 11. Вставить 10. Вставить 40. Вставить 30. Вставить 6. Вставить 4. Удалить 10. Вставить 3.

Выходные данные (с промежуточной информацией):

Обратите внимание : дерево выводится слева-направо.

Создано АВЛ-дерево с корнем.

Результаты тестирования представлены в таблице 1.

Таблица 1 — Тестирование.

№	Входные данные	Выходные данные
1.	1 35	1.
2.	1 11	-!-!-!-!-!-!-!-!-!-!-!-!- 35 11 -!-!-!-!-!-!-!-!-!-!-!-!-
3.	3.	-!-!-!-!-!-!-!-!-!-!-!-!- 35 11 10 -!-!-!-!-!-!-!-!-!-!-!-!-
4.	1 40	-!-!-!-!-!-!-!-!-!-!-!-!- 40 35 11 10 -!-!-!-!-!-!-!-!-!-!-!-!-
5.	1 30	-!-!-!-!-!-!-!-!-!-!-!-!- 40 35 30 11 10 -!-!-!-!-!-!-!-!-!-!-!-!-
6.	1 6	-!-!-!-!-!-!-!-!-!-!-!-!- 40

		35 30 11 10 6 -!-!-!-!-!-!-!-!-!-!-!-!-!-
7.	1 4	-!-!-!-!-!-!-!-!-!-!-!-!-!- 40 35 30 11 10 6 4 -!-!-!-!-!-!-!-!-!-!-!-!-!-
8.	2 10	-!-!-!-!-!-!-!-!-!-!-!-!-!- 40 35 30 11 10 6 -!-!-!-!-!-!-!-!-!-!-!-!-!-
9.	1 3	-!-!-!-!-!-!-!-!-!-!-!-!-!- 40 35 30 11 10 6 3 -!-!-!-!-!-!-!-!-!-!-!-!-!-

5. ИССЛЕДОВАНИЕ

5.1 Алгоритм вставки

Добавим ключ t. Будем спускаться по дереву, как при поиске ключа. Если находимся в вершине a, следовательно нужно идти в поддерево, если его нет , делаем t листом, а вершину a его корнем. Далее поднимаемся вверх по пути поиска и пересчитываем баланс у вершин. Если мы поднялись в вершину из левого поддерева, увеличиваем на единицу, если из правого, уменьшаем на единицу. Если пришли в вершину, её баланс стал равен нулю, значит высота поддерева не изменилась и подъём останавливается, если пришли в вершину и её баланс стал равным 1 или −1, значит высота поддерева изменилась и подъём продолжается. Если пришли в вершину, её баланс стал равным 2 или −2, делаем одно из четырёх вращений и, если после вращения баланс стал равным нулю, останавливаемся, иначе продолжаем подъём.

Можно сделать вывод, что при добавлении 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(log n)операций.

5.2. Алгоритм удалении

Если вершина — лист, то удалим её, если нет , найдём самую близкую по значению вершину , переместим её на место удаляемой вершины и удалим ее. От удалённой вершины будем подниматься вверх к корню и пересчитывать баланс у вершин. Если поднялись в вершину из левого поддерева, то уменьшаем значение вершины на единицу, если из правого, то увеличивается на единицу. Если баланс вершины равен 1 или −1, значит, высота этого поддерева не изменилась и подъём можно остановить. Если баланс вершины равен нулю, высота поддерева уменьшилась и подъём нужно продолжить. Если баланс баланс 2 или −2, следует выполнить одно из четырёх вращений и, если

после вращений баланс вершины стал равным нулю, то подъём продолжается, иначе останавливается.

Получаем , что на удаление вершины и балансировку суммарно тратится, O(h)операций. Таким образом, требуемое количество действий —O(log n).

5.3. ПЛАН ИССЛЕДОВАНИЯ

Был создан класс Research, который генерирует входные данные двух типов -строго возрастающей последовательности, и случайной. Каждая последовательность подается на вход операции вставки, потом генерируется набор индексов элементов для удаления. Этот набор подается на вход операции исключения. Во время работы фиксируется количество вызовов функций вставки , удаления.

5.4. РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ

Ниже приведены графикb, иллюстрирующие асимптотику выполнения операций в обоих случаях. На всех графиках оранжевой линией нарисован график логарифма от количества элементов в дереве. Этот график позволяет убедиться, что теоретическая оценка совпадает с практикой.

Визуализация была выполнена с помощью графиков, построенных с помощью python3.

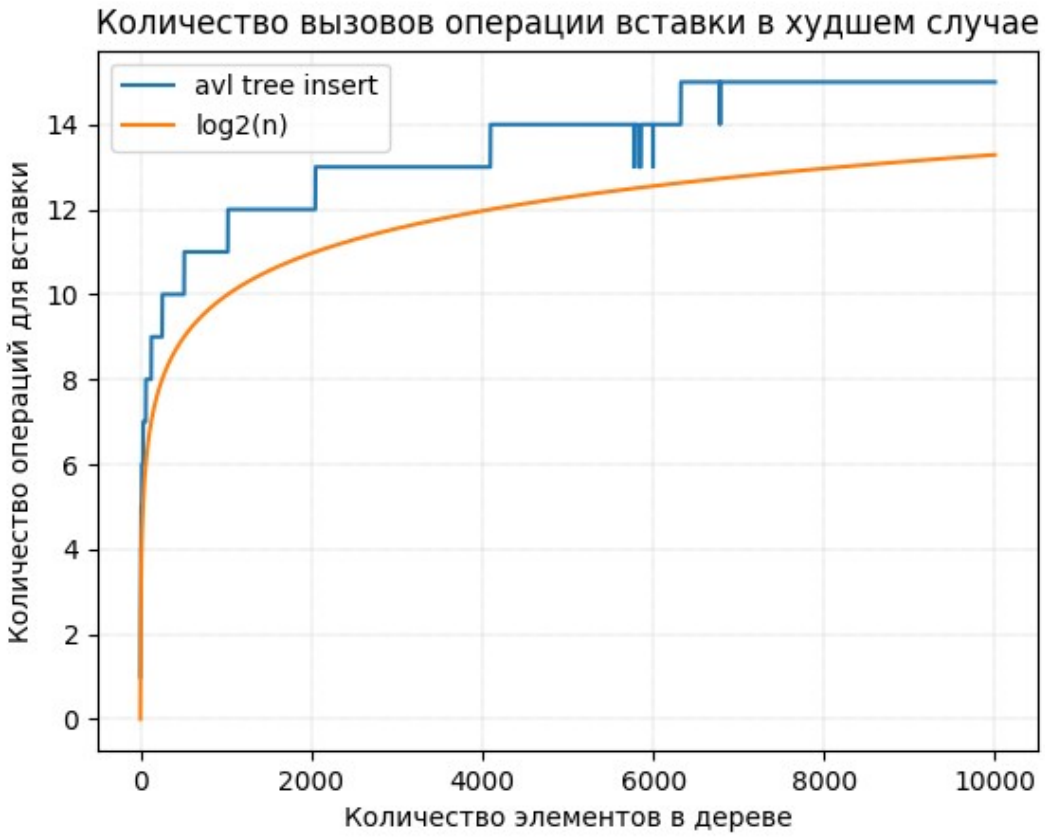

Рисунок 3. - Количество вызовов операции вставки в худшем случае.

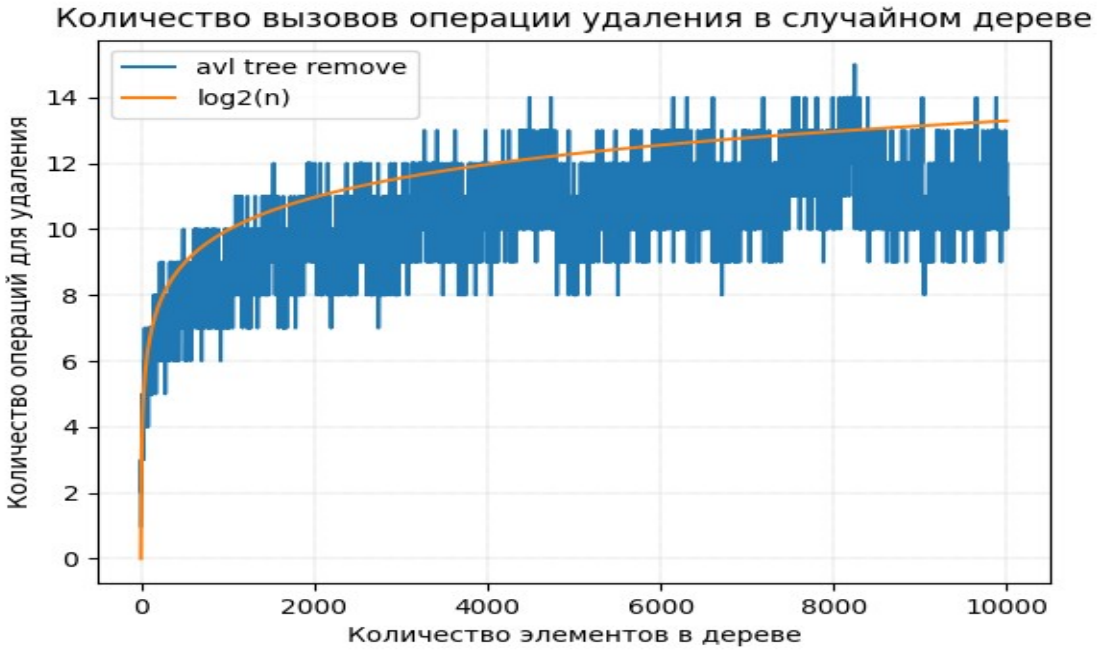

Рисунок 4. - Количество вызовов операции удаления в случайном дерево.

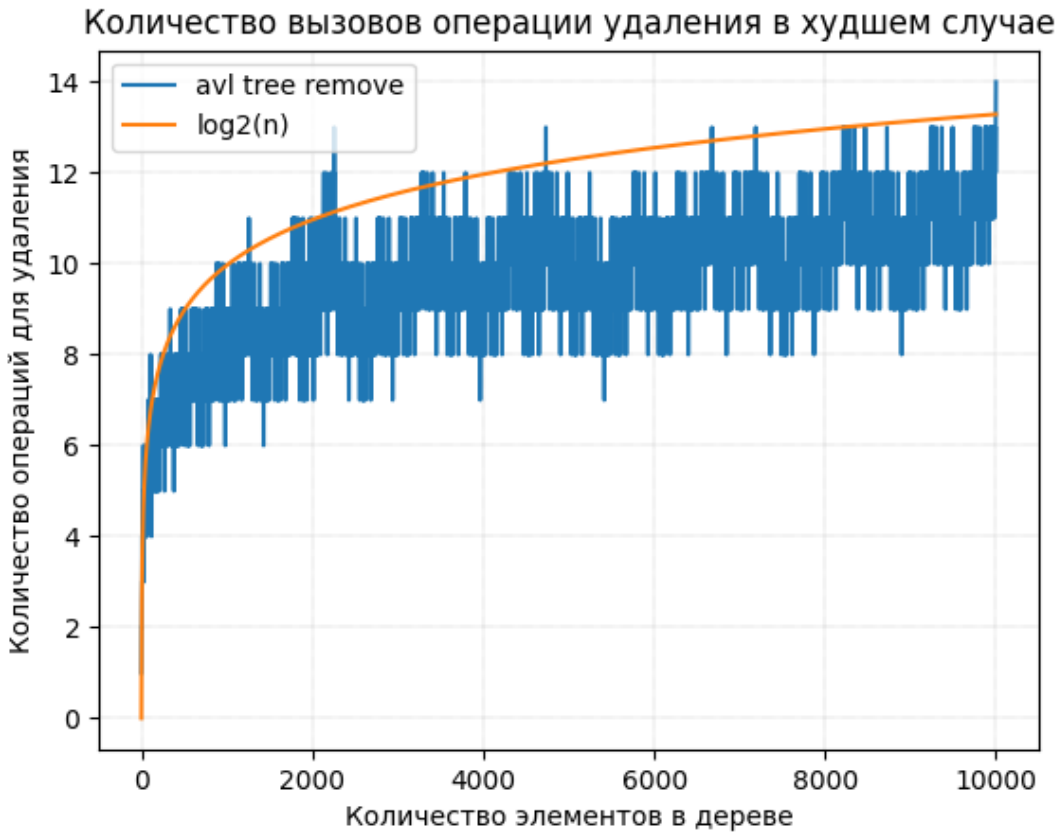

Рисунок 5. - Количество вызовов операции удаления в худшем случае .

ЗАКЛЮЧЕНИЕ

Была выполнена основная цель курсовой работы — исследован алгоритм вставки и исключения в АВЛ-деревьях.

Для реализации данной цели были решены следующие задачи:

1. Собраны теоретические сведения по изучаемой структуре данных и функциях для работы с ней.
2. Написана программа , реализующую заданную структуру данных и необходимые алгоритмы, а также провести исследование.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://monster-book.com/grokaem-algoritmy
2. https://habr.com/ru/post/150732/
3. https://monster-book.com/algoritmy-teoriya-i-prakticheskoe-primenenie

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp

```
#include <bits/stdc++.h>

using namespace std;
static int op_count = 0;
static int rot_count = 0;
class Node {
protected:
    int value; // значение в узле
    int height; // уровень узла
    Node *lt; // указатель на левого сына
    Node *rt; // указатель на правого сына
    Node *rotate_right(); // правое вращение
    Node *rotate_left(); // левое вращение

public:
    Node(int k) : value(k), lt(nullptr), rt(nullptr), height(1) {}
    int get_height(); // возвращает высоту
    int diff_height(); // возвращает разницу высот детей
    void update_height(); // возвращает новую высоту
    Node *balance(); // балансировка дерева
    Node *get_right(); // возвращает правого сына
    Node *get_left(); // возвращает левого сына
    void set_left(Node *node); // задает левого сына
    void set_right(Node *node); // задает правого сына
    void set_value(int value); // задает значение узла
    int get_value(); // возвращает значение
};

int Node::get_height() {
    return this->height;
}

int Node::diff_height() {
    if(this->rt == nullptr && this->lt == nullptr) return 0;
    else if(this->rt == nullptr) return (-1)*this->lt-
>get_height();
    else if(this->lt == nullptr) return this->rt->get_height();
    return this->rt->get_height() - this->lt->get_height();
}

void Node::update_height() {
    int hl = this->lt != nullptr ? this->lt->get_height() : 0;
    int hr = this->rt != nullptr ? this->rt->get_height() : 0;
    this->height = max(hl, hr) + 1;
}
```

```
Node *Node::rotate_right() {
    Node *new_root = this->lt;
    this->lt = new_root->rt;
    new_root->rt = this;
    this->update_height();
    new_root->update_height();
    return new_root;
}

Node *Node::rotate_left() {
    Node *new_root = this->rt;
    this->rt = new_root->lt;
    new_root->lt = this;
    this->update_height();
    new_root->update_height();
    return new_root;
}

Node *Node::balance() {
    rot_count++;
    this->update_height();
    int diff = this->diff_height();
    if (diff == 2) {
        if (this->rt != nullptr){
          if (this->rt->diff_height() < 0) this->rt = this->rt-
>rotate_right();
        }
        return this->rotate_left();
    } else if (diff == -2) {
        if (this->lt != nullptr){
          if (this->lt->diff_height() > 0) this->lt = this->lt-
>rotate_left();
        }
        return this->rotate_right();
    }
    return this;
}

Node *Node::get_right() {
    return this->rt ? this->rt : nullptr;
}

Node *Node::get_left() {
    return this->lt ? this->lt : nullptr;
}

int Node::get_value() {
    return this->value ? this->value : 0;
}

void Node::set_left(Node *node) {
    this->lt = node;
}
```

```
void Node::set_right(Node *node) {
    this->rt = node;
}

void Node::set_value(int value) {
    this->value = value;
}

class AVL_tree {
public:
    Node *root; // корень дерева
    AVL_tree() : root(nullptr) {};
    AVL_tree(int k);
    void print_tree(Node *node, int level); // печать дерева
    Node *insert_node(Node *node, int value); // вставка узла
    Node *find_min(Node *node); // найти минимальный узел
    Node *remove_min(Node *node); // удалить минимальный узел
    Node *remove_node(Node *node, int value); // удалить узел
    Node *lets_insert_node(Node *root, int value);
    Node *lets_remove_node(Node *root, int value);
};

AVL_tree::AVL_tree(int k) {
    cout << "[Created avl tree| root:" << k << "]\n\n";
    this->root = new Node(k);
    cout << "-!-!-!-!-!-!-!-!-!-!-!-!-!-" << endl;
    this->print_tree(this->root, 0);
    cout << "-!-!-!-!-!-!-!-!-!-!-!-!-!-" << endl;
}

void AVL_tree::print_tree(Node *node, int level) {
    if (node) {
        print_tree(node->get_right(), level + 1);
        for (int i = 0; i < level; i++) cout << "   ";
        cout << node->get_value() << endl;
        print_tree(node->get_left(), level + 1);
    }
}

Node *AVL_tree::insert_node(Node *node, int value) {
    op_count++;
    if (node == nullptr) return new Node(value);
    if (value < node->get_value()) {
        node->set_left(insert_node(node->get_left(), value));
    } else if (value > node->get_value()) {
        node->set_right(insert_node(node->get_right(), value));
    }
    return node->balance();
}

Node *AVL_tree::remove_node(Node *node, int value) {
    op_count++;
```

```
    if (node == nullptr) {
        return nullptr;
    }
    if (value < node->get_value()) {
        node->set_left(remove_node(node->get_left(), value));
    } else if (value > node->get_value()) {
        node->set_right(remove_node(node->get_right(), value));
    } else {
        Node *rt = node->get_right();
        Node *lt = node->get_left();
        delete node;
        if (!rt) return lt;
        Node *min = find_min(rt);
        min->set_right(remove_min(rt));
        min->set_left(lt);
        return min->balance();
    }
    return node->balance();
}

Node *AVL_tree::find_min(Node *node) {
    return node->get_left() ? find_min(node->get_left()) : node;
}

Node *AVL_tree::remove_min(Node *node) {
    op_count++;
    if (node->get_left() == nullptr) {
        return node->get_right();
    }
    node->set_left(remove_min(node->get_left()));
    return node->balance();
}

Node *AVL_tree::lets_insert_node(Node *root, int value) {
    cout << "[Insert element:" << value << "]\n\n";
    root = this->insert_node(root, value);
    cout << "-!-!-!-!-!-!-!-!-!-!-!-!-!-" << endl;
    this->print_tree(root, 0);
    cout << "-!-!-!-!-!-!-!-!-!-!-!-!-!-" << endl;
    return root;
}

Node *AVL_tree::lets_remove_node(Node *root, int value) {
    cout << "[Remove element:" << value << "]\n\n";
    root = this->remove_node(root, value);
    cout << "-!-!-!-!-!-!-!-!-!-!-!-!-!-" << endl;
    this->print_tree(root, 0);
    cout << "-!-!-!-!-!-!-!-!-!-!-!-!-!-" << endl;
    return root;
}

void print_menu() {
    cout << "1.Insert element\n"
```

```
            "2.Remove element\n"
            "3.Exit\n\n";
}

AVL_tree *process_user_input(AVL_tree *tree) {
    int f;
    print_menu();
    string user_value;
    cin >> f;
    switch (f) {
        case 1:
            cout << "Enter element: \n";
            cin >> user_value;
            try{
              if (tree) {
                tree->root = tree->lets_insert_node(tree->root,
stoi(user_value));
              } else {
                  tree = new AVL_tree(stoi(user_value));
              }
            } catch (...){
              cout << "Wrong Input" << endl << endl;
            }
            break;
        case 2:
            if (tree) {
                cout << "Enter element: \n";
                cin >> user_value;
                try{
                  tree->root = tree->lets_remove_node(tree->root,
stoi(user_value));
                } catch (...) {
                  cout << "Wrong Input" << endl << endl;
                }
            } else cout << "Tree is empty! \n";
            break;
        case 3:
            exit(0);
    }
    return tree;
}

class Research {
    int input_size;
public:
    unordered_set<int> input;
    Research(int v = 10000) : input_size(v) {};
    void generate_ascendance();

    void generate_random(int lower, int upper);

    void run_add(AVL_tree *tree, string str);
```

```
    void run_delete(AVL_tree *tree, string str);

};

void Research::generate_ascendance() {
    for(int i = 1; i <= input_size; i++) {
        input.insert(i);
    }
}

void Research::generate_random(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);
    input.clear();
    while(input.size() < input_size) {
        input.insert(distribution(gen));
    }
}


void Research::run_add(AVL_tree *tree, string str) {
    int tree_size = 0;
    ofstream out;
    out.open(str);
    out << "tree_size," << "op_count," << "rot_count" << endl;
    for(auto x : this->input) {
        op_count = 0;
        rot_count = 0;
        tree_size++;
        tree->root = tree->insert_node(tree->root, x);
        out << tree_size << ',' << op_count << "," << rot_count <<
"\n";
    }
    out.close();
}

void Research::run_delete(AVL_tree *tree, string str) {
    ofstream out;
    int tree_size = input_size;
    out.open(str);
    out << "tree_size," << "op_count," << "rot_count" << endl;
    for(auto index : input) {
        op_count = 0;
        rot_count = 0;
        tree->root = tree->remove_node(tree->root, tree->root-
>get_value());
        out << tree_size << ',' << op_count  << "," << rot_count
<< "\n";
        tree_size--;
    }
    out.close();
```

```
}

int main() {
  AVL_tree *tree = new AVL_tree();
  Research res;
  int f;
  string user_value;
  cout << "1.Interactive tree\n"
          "2.Research tree\n"
          "3.Exit\n\n";
  cin >> f;
  switch (f) {
    case 1:
        while (true) {
          tree = process_user_input(tree);
        }
        break;
    case 2:
        cout << "Generation of consecutive elements..." << endl;
        res.generate_ascendance();
        cout << "Research add..." << endl;
        res.run_add(tree, "research_add.csv");
        cout << "Recorded in the file research_add.csv" << endl;
        cout << "Research delete..." << endl;
        res.run_delete(tree, "research_delete.csv");
        cout << "Recorded in the file research_delete.csv" <<
endl;
        cout << "Generating random elements..." << endl;
        res.generate_random(0, 10000);
        cout << "Research add..." << endl;
        res.run_add(tree, "research_add_random.csv");
        cout << "Recorded in the file research_add_random.csv" <<
endl;
        cout << "Research delete..." << endl;
        res.run_delete(tree, "research_delete_random.csv");
        cout << "Recorded in the file research_delete_random.csv"
<< endl;
        break;
    case 3:
        exit(0);
  }
  return 0;
}
```

grafiki.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
lg = {
'count': [x for x in range(1, 10000, 1)],
'val': [np.log2(x) for x in range(1, 10000, 1)]
}
df1 = pd.read_csv('research_add.csv')
df2 = pd.read_csv('research_delete.csv')
df3 = pd.read_csv('research_add_random.csv')
df4 = pd.read_csv('research_delete_random.csv')

fig = plt.figure()

fig.set_figheight(10)
fig.set_figwidth(20)
fig.subplots_adjust()
ax1 = fig.add_subplot(221)
ax2 = fig.add_subplot(222)
ax3 = fig.add_subplot(223)
ax4 = fig.add_subplot(224)



ax1.title.set_text('Количество вызовов операции вставки в худшем
случае')
ax1.set_xlabel('Количество элементов в дереве')
ax1.set_ylabel('Количество операций для вставки')
ax1.plot(df1['tree_size'], df1['op_count'])
ax1.plot(lg['count'], lg['val'])
ax1.grid(color='gray', linestyle=':', linewidth=0.3)
ax1.legend(['avl tree insert', 'log2(n)'], loc=2)

ax3.title.set_text('Количество вызовов операции вставки в
случайном дереве')
ax3.set_xlabel('Количество элементов в дереве')
ax3.set_ylabel('Количество операций для вставки')
ax3.plot(df3['tree_size'], df3['op_count'])
ax3.plot(lg['count'], lg['val'])
ax3.grid(color='gray', linestyle=':', linewidth=0.3)
ax3.legend(['avl tree insert random', 'log2(n)'], loc=2)

ax2.title.set_text('Количество вызовов операции удаления в худшем
случае')
ax2.set_xlabel('Количество элементов в дереве')
ax2.set_ylabel('Количество операций для вставки')
ax2.plot(df2['tree_size'], df2['op_count'])
ax2.plot(lg['count'], lg['val'])
ax2.grid(color='gray', linestyle=':', linewidth=0.3)
ax2.legend(['avl tree remove', 'log2(n)'], loc=2)

ax4.title.set_text('Количество вызовов операции удаления в
случайном дереве')
```

```
ax4.set_xlabel('Количество элементов в дереве')
ax4.set_ylabel('Количество операций для вставки')
ax4.plot(df4['tree_size'], df4['op_count'])
ax4.plot(lg['count'], lg['val'])
ax4.grid(color='gray', linestyle=':', linewidth=0.3)
ax4.legend(['avl tree remove random', 'log2(n)'], loc=2)

plt.savefig("research.png")
```