

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 AVL-дерева

Студент гр. 9384

Соседков К.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Студент Соседков К.С.

Группа 9384

Тема работы : Исследование AVL-дерева

Исходные данные:

Генерация входных данных, использование их для измерения количественных характеристик структур данных, алгоритмов, действий, сравнение экспериментальных результатов с теоретическими

Содержание пояснительной записки:

«Содержание» «Введение» «Анализ задачи» «Визуализация»

«Сопоставление с теоретическими оценками» «Тестирование» «Заключение»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 15.11.2000

Дата сдачи реферата: 24.12.2000

Дата защиты реферата: 24.12.2000

Студент

Соседков К.С.

Преподаватель

Ефремов М.А.

АННОТАЦИЯ

Основное содержание данного курсового проекта — исследование методов вставки и удаления из структуры данных АВЛ-дерево. Для выполнения данной работы подсчитывается количество операций для каждого из методов и полученная статистика отображается в виде графиков на экран. Анализируя полученные графики можно убедиться что сложность данных методов - логарифмическая, что полностью соответствует теоретическим оценкам.

SUMMARY

The main content of this course project is the study of methods of insertion and deletion from the data structure AVL-tree. To perform this work, the number of operations for each of the methods is calculated and the statistics obtained are displayed in the form of graphs on the screen. Analyzing the graphs obtained, one can make sure that the complexity of these methods is logarithmic, which fully corresponds to theoretical estimates.

СОДЕРЖАНИЕ

	Введение	5
1.	Анализ задачи	5
1.1.	Основные теоретические сведения	5
1.2.	План экспериментального исследования	6
1.3	Генерация данных и накопление статистики	7
2.	Визуализация	7
3.	Сопоставление с теоретическими оценками	9
4.	Тестирование	10
	Заключение	12
	Список использованных источников	14
	Приложение А. Исходный код программы	15

ВВЕДЕНИЕ

Целью данной работы является исследование методов вставки и удаления из структуры данных AVL-дерево, а так же сравнение полученных результатов с теоретическими.

1. АНАЛИЗ ЗАДАЧИ

1.1. Основные теоретические сведения

Целью курсовой работы является исследование алгоритмов вставки и исключения в AVL-дерево в среднем и в худшем случае. AVL-дерево — это сбалансированное по высоте бинарное дерево поиска, для каждой его вершины высота двух его поддеревьев различается не более чем на единицу.

В первую очередь для выполнения работы был разработан шаблонный класс `AVL_Tree` со следующими методами:

`insert` — вставка элемента в дерево

`remove` — удаление элемента из дерева

`rotate_left` — левый поворот дерева

`rotate_right` — правый поворот дерева

Методы `insert` и `remove` аналогичны тем же методам в обычном бинарном дереве поиска, за исключением того что после вставки или удаления нужно проверять не нарушился ли баланс дерева или его поддеревьев. Вставка и удаление имеют логарифмическую сложность.

Методы `rotate_left` и `rotate_right` имеют спецификатор доступа `private`, они вызываются когда баланс дерева нарушается ($\text{abs}(t.\text{left}-t.\text{right}) \geq 2$). Баланс AVL-дерева может нарушиться только после вставки и удаления.

Всего существует четыре вида поворота, они являются комбинациями этих двух:

LL-поворот: поворот дерева влево

LR-поворот: поворот правого поддерева вправо, затем поворот дерева влево

RR-поворот: поворот дерева вправо

RL-поворот: поворот левого поддерева влево, затем поворот дерева вправо

Повороты работают за константное время.

В методах AVL-дерева были реализованы необходимые счетчики фиксирующие количество операций необходимых для выполнения этих методов. Для получения количества операций был разработан метод `number_of_operations`.

1.2. План экспериментального исследования

Для исследования методов вставки и удаления были выполнены следующие шаги:

1. Реализация структуры данных AVL-дерево
2. Генерация данных
3. Фиксация результатов испытаний алгоритмов
4. Визуализация полученных результатов в виде графиков

1.3 Генерация данных и накопление статистики

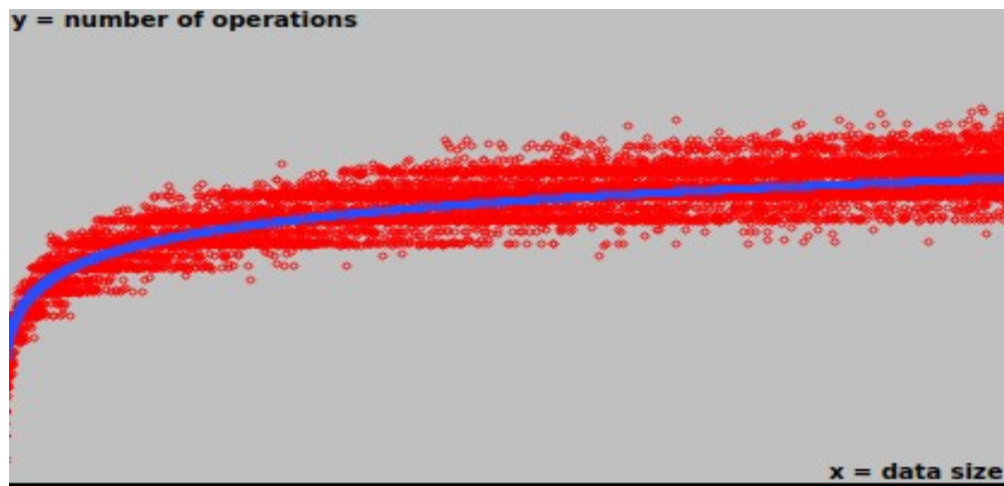
В данной работе используются два вида данных, случайные данные и отсортированные. Данные для исследования генерируются с помощью класса Qt - `QRandomGenerator`. Результат работы алгоритмов с этими данными сразу же накладывается на графики(подробнее в следующем разделе).

2. ВИЗУАЛИЗАЦИЯ

Визуализация полученной статистики выполняется с помощью классов `QGraphicsScene` и `QGraphicsView` фреймворка Qt. Всего показывается четыре графика, по два на каждый метод. Один их графиков показывает статистику со случайными данными, другой с отсортированными. Ось x — количество элементов в AVL-дерево, ось y — количество операций необходимых для вставки(удаления) в AVL-дерево размером x. Так же показывается полное время выполнения алгоритмов.

Пример графика изображающего удаление из AVL-дерева размера 10000 показан на изображении 1. Красные точки показывают количество операций. Синий график — $17\log(x)$. Время удаления всех элементов из этого дерева — 19 мс.

Изображение 1.

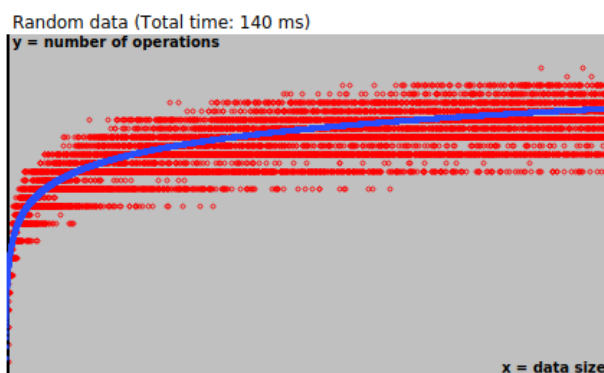


3. СОПОСТАВЛЕНИЕ С ТЕОРЕТИЧЕСКИМИ ОЦЕНКАМИ

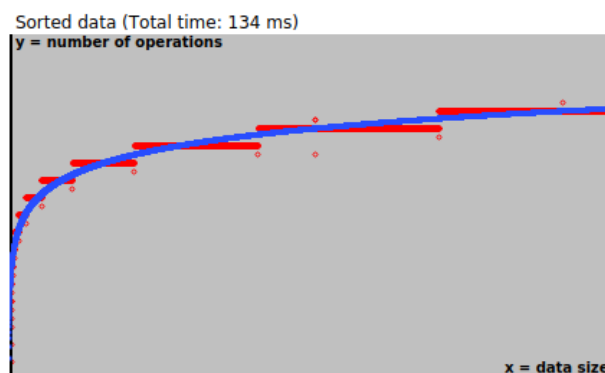
На изображении 2 представлен результат работы программы.

Изображение 2. Количество данных равно 10000.

Insert statistics

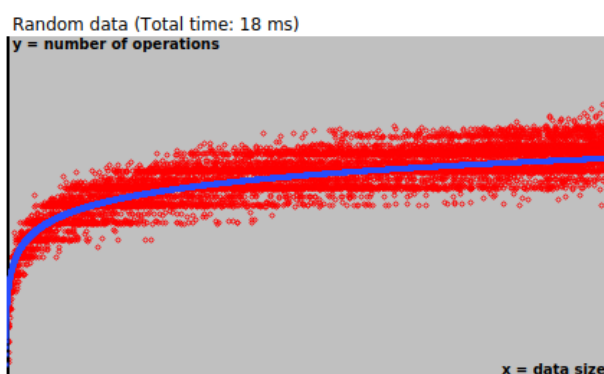


$x \in [0, 10000]$ $y \in [0, 200]$ \blacksquare $17\log(x)$
 \blacksquare number of operations(random data)

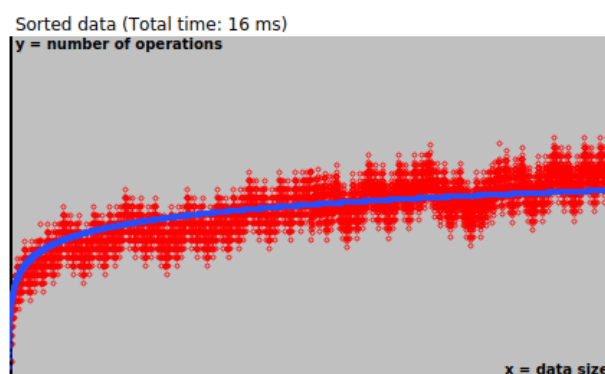


$x \in [0, 10000]$ $y \in [0, 200]$ \blacksquare $17\log(x)$
 \blacksquare number of operations(sorted data)

Remove statistics



$x \in [0, 10000]$ $y \in [0, 200]$ \blacksquare $12\log(x)$
 \blacksquare number of operations(random data)



$x \in [0, 10000]$ $y \in [0, 200]$ \blacksquare $12\log(x)$
 \blacksquare number of operations(sorted data)

На изображении 2 видно что методы вставки и удаления из AVL-дерева имеют логарифмическую сложность($O(\log(n))$). Причем почти для любых размеров дерева вставка и удаления случайных элементов требуют большего числа операций чем для отсортированных данных. Это связано с тем что дерево будет заполняться не равномерно, и в среднем случае балансировка дерева будет выполняться чаще. Ну может так получится что сгенерированные данные будут лучше чем отсортированные, но вероятность этого очень мала. Лучшим случаем является равномерное заполнение(удаление) дерева.

Так же из графиков видно что хотя сложность алгоритмов вставки и удаления одинакова, удаление все равно работает в несколько раз быстрее. Связано это с тем что при удалении балансировка выполняется не так часто как при вставке. Удаление из АВЛ-дерева $\sim 12 \cdot \log(n)$, вставка $\sim 17 \cdot \log(n)$. Коэффициенты 12 и 17 не точны, они зависят от способа подсчета количества операций.

4. ТЕСТИРОВАНИЕ

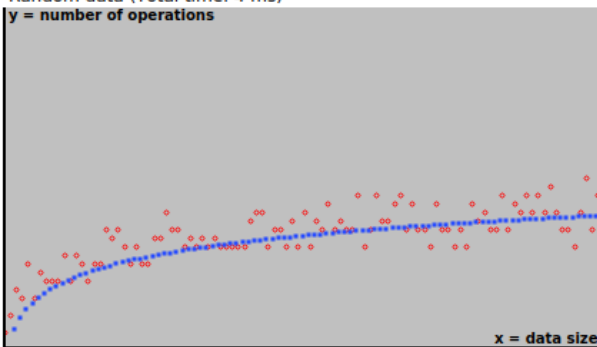
Ниже представлены изображения показывающие результат работы программы при вводе различных размеров дерева.

Изображение 3. Количество данных равно 100.

Insert statistics

Random data (Total time: 4 ms)

y = number of operations

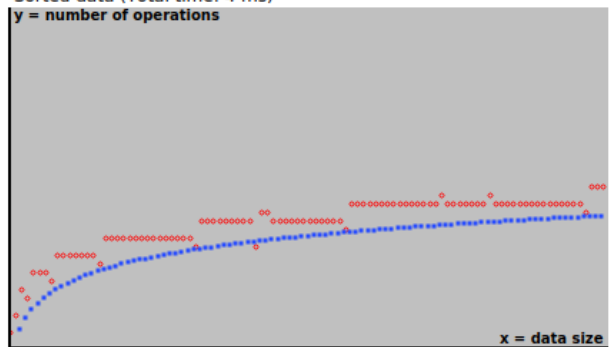


x ∈ [0,100] ■ 17log(x)

y ∈ [0,200] ■ number of operations(random data)

Sorted data (Total time: 4 ms)

y = number of operations



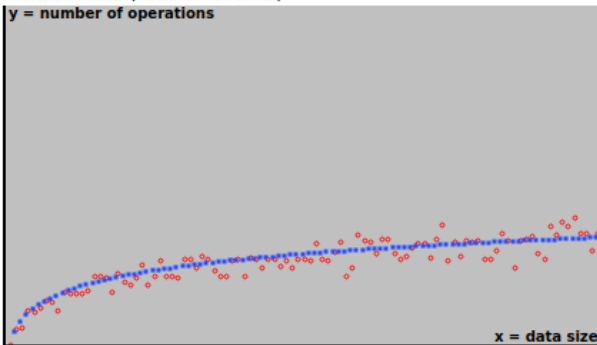
x ∈ [0,100] ■ 17log(x)

y ∈ [0,200] ■ number of operations(sorted data)

Remove statistics

Random data (Total time: 0 ms)

y = number of operations

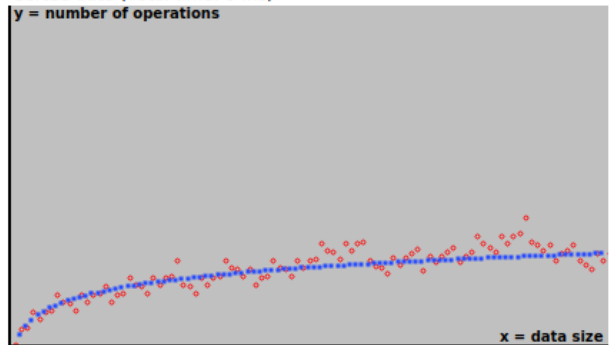


x ∈ [0,100] ■ 12log(x)

y ∈ [0,200] ■ number of operations(random data)

Sorted data (Total time: 0 ms)

y = number of operations

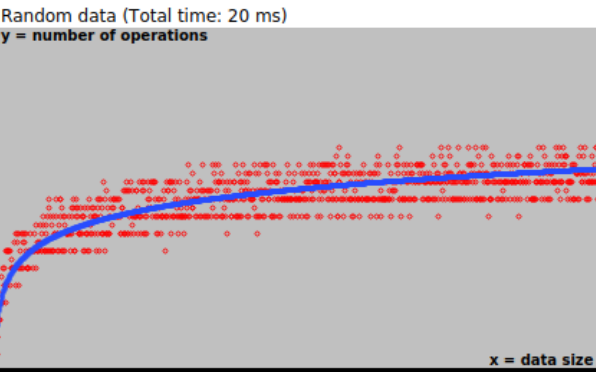


x ∈ [0,100] ■ 12log(x)

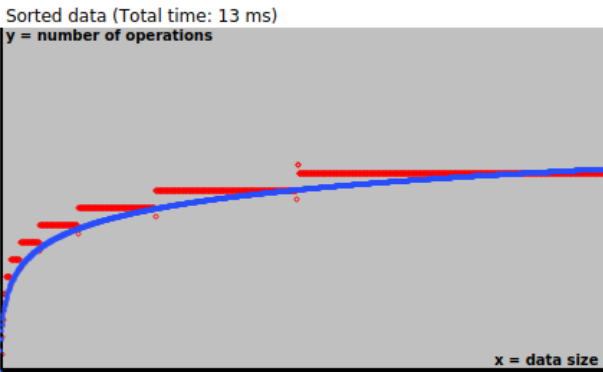
y ∈ [0,200] ■ number of operations(random data)

Изображение 4. Количество данных равно 1000.

Insert statistics

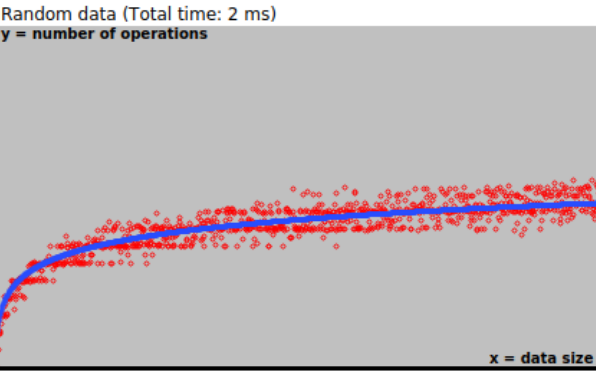


$x \in [0,1000]$ ■ $17\log(x)$
 $y \in [0,200]$ ■ number of operations(random data)

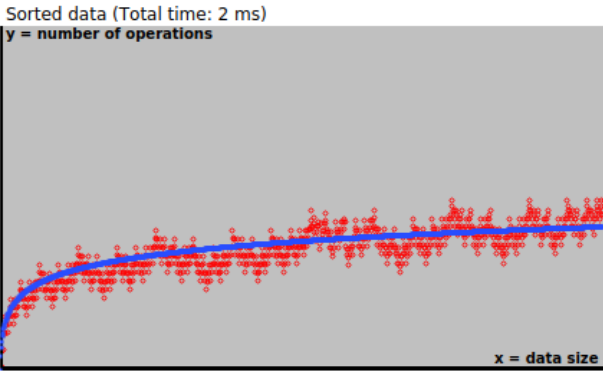


$x \in [0,1000]$ ■ $17\log(x)$
 $y \in [0,200]$ ■ number of operations(sorted data)

Remove statistics



$x \in [0,1000]$ ■ $12\log(x)$
 $y \in [0,200]$ ■ number of operations(random data)



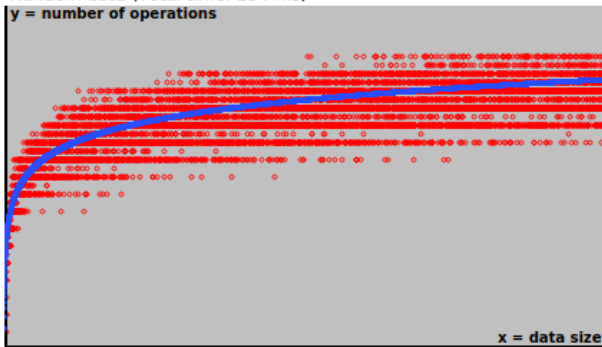
$x \in [0,1000]$ ■ $12\log(x)$
 $y \in [0,200]$ ■ number of operations(random data)

Generate

Изображение 5. Количество данных равно 10000.

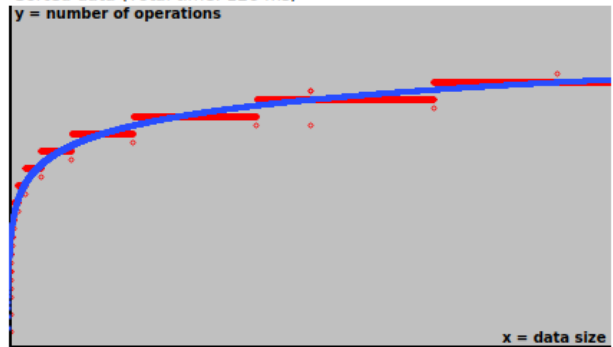
Insert statistics

Random data (Total time: 134 ms)



$x \in [0, 10000]$ $y \in [0, 200]$
■ $17\log(x)$ ■ number of operations(random data)

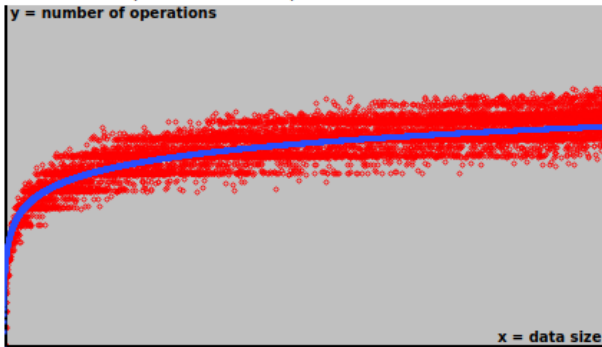
Sorted data (Total time: 126 ms)



$x \in [0, 10000]$ $y \in [0, 200]$
■ $17\log(x)$ ■ number of operations(sorted data)

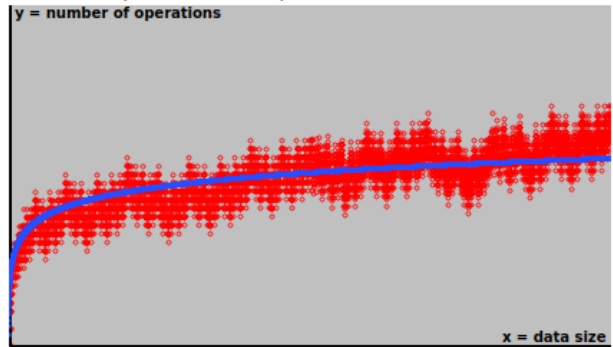
Remove statistics

Random data (Total time: 17 ms)



$x \in [0, 10000]$ $y \in [0, 200]$
■ $12\log(x)$ ■ number of operations(random data)

Sorted data (Total time: 16 ms)



$x \in [0, 10000]$ $y \in [0, 200]$
■ $12\log(x)$ ■ number of operations(sorted data)

ЗАКЛЮЧЕНИЕ

При выполнении работы были исследованы методы вставки и удаления из АВЛ-дерева, и так же в виде графиков была изображена накопленная статистика. В результате исследования было показано что методы вставки и удаления из АВЛ-дерева имеют логарифмическую сложность, что в точности соответствует теоретическим оценкам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы: построение и анализ : пер. с англ. / Т. Кормен и др. – 2-е изд. – М. : Издательский дом «Вильямс», 2007, 2009

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: AVL_Tree.h

```
#ifndef AVL_TREE_H
#define AVL_TREE_H

#include <QDebug>
#include <QTime>
template <class T>
class AVL_Tree
{
public:
    struct AvlNode {
        AvlNode(T data) {
            this->data = data;
        }

        ~AvlNode() {
            if(left) delete left;
            if(right) delete right;
        }

        AvlNode* leftChild() const {
            return this->left;
        }

        AvlNode* rightChild() const {
            return this->right;
        }

        int balance_factor() {
            int left_height = left ? left->height : 0;
            int right_height = right ? right->height : 0;
            return left_height-right_height;
        }

        void update_height() {
            int left_henght = left ? left->height : 0;
            int right_henght = right ? right->height : 0;
            height = std::max(left_henght,right_henght)+1;
        }

        void print(int offset=0) {
            qDebug() << QString(" ").repeated(offset) << this->data << this->height;
            if(left) left->print(offset+2);
            if(right) right->print(offset+2);
        }
    };
};
```

```

    T data;
    int height = 1;
    AvlNode* left = nullptr;
    AvlNode* right = nullptr;

};

AVL_Tree() {};

~AVL_Tree() {
    if(root) delete root;
};

int number_of_operations() const {
    return num_operations;
}

int elapsed_time() const {
    return m_seconds;
}

void clear() {
    if(root) {
        delete root;
        root = nullptr;
    }
}

void insert(T data) {
    num_operations = 0;
    QTime t;
    t.start();
    root = insert(data, root);
    m_seconds = t.elapsed();
}

void remove(T data) {
    num_operations = 0;
    root = remove(data, root);
}

int height() {
    if(!root) return 0;
    return root->height;
}

void print(int offset=0) {

```



```

    this->root->print();
}

```

```

AvlNode* getRoot() const {
    return root;
}

```

private:

```

AvlNode* insert(T data, AvlNode* node) {
    if(!node) {
        num_operations+=2;
        node = new AvlNode(data);
    }
    else if(data > node->data) {
        num_operations++;
        node->right = insert(data, node->right);
    }
    else if(data < node->data) {
        num_operations++;
        node->left = insert(data, node->left);
    }
    num_operations++;
    return balance(node);
}

```

```

AvlNode* remove(T data, AvlNode* node) {
    if(!node) {
        num_operations++;
        return node;
    }
    else if(data > node->data) {
        num_operations++;
        node->right = remove(data, node->right);
    }
    else if(data < node->data) {
        num_operations++;
        node->left = remove(data, node->left);
    }
    else {
        if(!node->left && !node->right) {
            num_operations+=3;
            delete node;
            node = nullptr;
            return nullptr;
        }
        else if(!node->left || !node->right) {
            num_operations+=3;
            if(node->left) {
                node->data = node->left->data;
                delete node->left;
                node->left = nullptr;
            }
        }
    }
}

```

```

        else {
            node->data = node->right->data;
            delete node->right;
            node->right = nullptr;
        }
    }
    else {
        num_operations+=2;
        AvlNode* tmp = min(node->right);
        node->data = tmp->data;
        node->right = remove(node->data, node->right);
    }
}
num_operations++;
return balance(node);
}

```

```

AvlNode* min(AvlNode* node) {
    AvlNode* tmp = node;
    while (tmp->left) {
        num_operations++;
        tmp = tmp->left;
    }
    return tmp;
}

```

```

AvlNode* balance(AvlNode* node) {
    if(!node) return node;
    node->update_height();
    num_operations+=3;
    if(node->balance_factor() >= 2) {
        if(node->left->balance_factor() <= -1) {
            num_operations+=5;
            node->left = rotate_left(node->left);
        }
        node = rotate_right(node);
        num_operations+=5;
    }
    else if(node->balance_factor() <= -2) {
        if(node->right->balance_factor() >= 1) {
            num_operations+=5;
            node->right = rotate_right(node->right);
        }
        node = rotate_left(node);
        num_operations+=5;
    }

    node->update_height();
    num_operations+=5;
    return node;
}

```

```

AvlNode* rotate_right(AvlNode* node) {
    AvlNode* new_root = node->left;
    node->left = new_root->right;
    new_root->right = node;
    new_root->right->update_height();
    return new_root;
}

```

```

AvlNode* rotate_left(AvlNode* node) {
    AvlNode* new_root = node->right;
    node->right = new_root->left;
    new_root->left = node;
    new_root->left->update_height();
    return new_root;
}

```

```

AvlNode* root = nullptr;
int num_operations = 0;
int m_seconds = 0;

```

```

};

```

```

#endif // AVL_TREE_H

```

Название файла: GraphicWidget.cpp

```

#include "graphicwidget.h"
#include <QtMath>
#include <QPainter>
#include <QGraphicsPixmapItem>
#include "AVL_Tree.h"
#include <QRandomGenerator>
#include <QVector>
#include <QTime>
#include <QGraphicsTextItem>
#include <QGraphicsProxyWidget>

```

```

GraphicWidget::GraphicWidget(QWidget *parent) : QWidget(parent) {

```

```

    scene = new QGraphicsScene(parent);
    view = new QGraphicsView(parent);
    view->resize(parent->width(), parent->height());
    view->setScene(scene);
    // view->setAlignment(Qt::AlignTop);
    scene->setSceneRect(view->contentsRect());

```

```

}

void GraphicWidget::clear() {
    scene->clear();
}

void GraphicWidget::drawStatistics(int data_size) {
    int insert_width = 420;
    int insert_height = 240;
    int x_max = data_size;
    int y_max = 200;
    int random_insertion_time;
    int sorted_insertion_time;
    int random_remove_time;
    int sorted_remove_time;
    QTime q;
    q.start();
    QPixmap insert_operations_pixmap = createPixmapGraph(insert_width, insert_height, x_max,
y_max, 17);
    QPixmap insert_operations_sorted_pixmap = insert_operations_pixmap.copy();
    QPixmap remove_operations_pixmap = createPixmapGraph(insert_width, insert_height, x_max,
y_max, 12);
    QPixmap remove_operations_sorted_pixmap = remove_operations_pixmap.copy();
    qDebug() << "create time: " << q.elapsed();

```

```

//RANDOM INSERT STATISTICS
QPainter p(&insert_operations_pixmap);
QPen pen(QColor(255,0,0));
p.setRenderHint(QPainter::Antialiasing, true);
p.setPen(pen);
AVL_Tree<int> tree;
QVector<quint32> data(x_max);
QRandomGenerator::global()->fillRange(data.data(), data.size());
QTime t;
t.start();
for(int i=0; i<data.size(); i++) {
    tree.insert(data[i]);
    float x = i;
    float y = tree.number_of_operations();
    if(x*insert_width/(x_max) > insert_width || (-y*insert_height/y_max)+insert_height >
insert_height) {
        break;
    }
    p.drawEllipse(x*insert_width/(x_max), (-y*insert_height/y_max)+insert_height, 3, 3);

```

```

}

qDebug() << "Random insert time (ms) " << t.elapsed();
random_insertion_time = t.elapsed();
p.end();


//RANDOM REMOVE STATISTICS
QPainter random_remove_p(&remove_operations_pixmap);
random_remove_p.setRenderHint(QPainter::Antialiasing, true);
random_remove_p.setPen(QColor(255,0,0));
t.start();
for(int i=0; i<data.size(); i++) {
    tree.remove(data[i]);
    float x = i;
    float y = tree.number_of_operations();
    if(x*insert_width/(x_max) > insert_width || (-y*insert_height/y_max)+insert_height >
insert_height) {
        break;
    }
    random_remove_p.drawEllipse(insert_width-x*insert_width/(x_max),
(-y*insert_height/y_max)+insert_height, 3, 3);
}

qDebug() << "Random remove time (ms) " << t.elapsed();
random_remove_time = t.elapsed();
random_remove_p.end();

```

```

//SORTED INSERT STATISTICS
std::sort(data.begin(), data.end());
QPainter p2(&insert_operations_sorted_pixmap);
QPen pen2(QColor(255,0,0));
p2.setRenderHint(QPainter::Antialiasing, true);
p2.setPen(pen2);

```

```

t.start();
for(int i=0; i<data.size(); i++) {
    tree.insert(data[i]);
    float x = i;
    float y = tree.number_of_operations();
    if(x*insert_width/(x_max) > insert_width || (-y*insert_height/y_max)+insert_height >
insert_height) {
        break;
    }
    p2.drawEllipse(x*insert_width/(x_max), (-y*insert_height/y_max)+insert_height, 3, 3);
}
qDebug() << "Sorted insert time (ms) " << t.elapsed();
sorted_insertion_time = t.elapsed();
p2.end();

```

```

//SORTED REMOVE STATISTICS
QPainter sorted_remove_p(&remove_operations_sorted_pixmap);
sorted_remove_p.setRenderHint(QPainter::Antialiasing, true);
sorted_remove_p.setPen(QColor(255,0,0));
t.start();
for(int i=0; i<data.size(); i++) {
    tree.remove(data[i]);
    float x = i;
    float y = tree.number_of_operations();
    if(x*insert_width/(x_max) > insert_width || (-y*insert_height/y_max)+insert_height >
insert_height) {
        break;
    }
    sorted_remove_p.drawEllipse(insert_width-x*insert_width/(x_max), (-y*insert_height/y_max)
+insert_height, 3, 3);
}

qDebug() << "Sorted remove time (ms) " << t.elapsed();
sorted_remove_time = t.elapsed();
sorted_remove_p.end();

```

```

insert_operations_pixmap = drawLog(insert_operations_pixmap, insert_width,insert_height,
x_max, y_max, 17);

```

```

remove_operations_pixmap = drawLog(remove_operations_pixmap, insert_width, insert_height,
x_max, y_max, 14);
insert_operations_sorted_pixmap = drawLog(insert_operations_sorted_pixmap,
insert_width, insert_height, x_max, y_max, 17);
remove_operations_sorted_pixmap = drawLog(remove_operations_sorted_pixmap,
insert_width, insert_height, x_max, y_max, 12);

scene->addPixmap(insert_operations_pixmap)->setPos(10,60);
scene->addPixmap(remove_operations_pixmap)->setPos(10,insert_height+190);
scene->addPixmap(insert_operations_sorted_pixmap)->setPos(50+insert_width,60);
scene->addPixmap(remove_operations_sorted_pixmap)-
>setPos(50+insert_width,insert_height+190);

```

```

//INSERT TEXT
QFont titleFont;
titleFont.setPixelSize(25);
titleFont.setBold(true);
titleFont.setFamily("Calibri");

```

```

QFont textFont;
textFont.setPixelSize(12);
textFont.setBold(false);
textFont.setFamily("Calibri");

```

```

QGraphicsTextItem* text = scene->addText("Insert statistics");
text->setPos(10,0);
text->setFont(titleFont);

```

```

text = scene->addText(QString("Random data ") + QString("(Total time: ") +
QString::number(random_insertion_time) + QString(" ms)"));
text->setPos(10,40);
text->setFont(textFont);

```

```

text = scene->addText(QString("Sorted data ") + QString("(Total time: ") +
QString::number(sorted_insertion_time) + QString(" ms)"));
text->setPos(insert_width+50, 40);
text->setFont(textFont);

```

```

text = scene->addText(QString("x ∈ [0,") + QString::number(x_max) + QString("]"));
text->setPos(10,insert_height+70);
text->setFont(textFont);

```

```

text = scene->addText(QString("y ∈ [0,") + QString::number(y_max) + QString("]"));
text->setPos(10,insert_height+90);
text->setFont(textFont);

```

```

QPixmap rect_pix(10,10);
rect_pix.fill(QColor(0,0,255));
scene->addPixmap(rect_pix)->setPos(90, insert_height+75);
rect_pix.fill(QColor(255,0,0));
scene->addPixmap(rect_pix)->setPos(90, insert_height+95);

```

```

text = scene->addText(QString("17log(x)"));
text->setPos(100, insert_height+70);
text->setFont(textFont);

```

```

text = scene->addText(QString("number of operations(random data)"));
text->setPos(100, insert_height+90);
text->setFont(textFont);

```

```

text = scene->addText(QString("x ∈ [0,") + QString::number(x_max) + QString("]"));
text->setPos(insert_width+50,insert_height+70);
text->setFont(textFont);

```

```

text = scene->addText(QString("y ∈ [0,") + QString::number(y_max) + QString("]"));
text->setPos(insert_width+50,insert_height+90);
text->setFont(textFont);

```

```

rect_pix.fill(QColor(0,0,255));
scene->addPixmap(rect_pix)->setPos(insert_width+50+80, insert_height+75);
rect_pix.fill(QColor(255,0,0));
scene->addPixmap(rect_pix)->setPos(insert_width+50+80, insert_height+95);

```

```

text = scene->addText(QString("17log(x)"));
text->setPos(insert_width+50+90, insert_height+70);
text->setFont(textFont);

```



```

text = scene->addText(QString("number of operations(sorted data)"));
text->setPos(insert_width+50+90, insert_height+90);
text->setFont(textFont);

```

```

//REMOVE TEXT
text = scene->addText("Remove statistics");
text->setPos(10,insert_height+130);
text->setFont(titleFont);

```

```

text = scene->addText(QString("Random data ") + QString("(Total time: ") +
QString::number(random_remove_time) + QString(" ms)"));
text->setPos(10,insert_height+170);
text->setFont(textFont);

```

```

text = scene->addText(QString("Sorted data ") + QString("(Total time: ") +
QString::number(sorted_remove_time) + QString(" ms)"));
text->setPos(insert_width+50, insert_height+170);
text->setFont(textFont);

```

```

text = scene->addText(QString("x ∈ [0,") + QString::number(x_max) + QString("]"));
text->setPos(10, 2*insert_height+170+30);
text->setFont(textFont);

```

```

text = scene->addText(QString("y ∈ [0,") + QString::number(y_max) + QString("]"));
text->setPos(10, 2*insert_height+170+30+20);
text->setFont(textFont);

```

```

rect_pix.fill(QColor(0,0,255));
scene->addPixmap(rect_pix)->setPos(90, 2*insert_height+170+30+5);
rect_pix.fill(QColor(255,0,0));
scene->addPixmap(rect_pix)->setPos(90, 2*insert_height+170+30+20+5);

```

```

text = scene->addText(QString("12log(x)"));
text->setPos(100, 2*insert_height+170+30);
text->setFont(textFont);

```

```

text = scene->addText(QString("number of operations(random data)"));
text->setPos(100, 2*insert_height+170+30+20);
text->setFont(textFont);

```

```

text = scene->addText(QString("x ∈ [0,") + QString::number(x_max) + QString("]"));
text->setPos(insert_width+50, 2*insert_height+170+30);
text->setFont(textFont);

```

```

text = scene->addText(QString("y ∈ [0,") + QString::number(y_max) + QString("]"));
text->setPos(insert_width+50, 2*insert_height+170+30+20);
text->setFont(textFont);

```

```

rect_pix.fill(QColor(0,0,255));
scene->addPixmap(rect_pix)->setPos(insert_width+50+80, 2*insert_height+170+30+5);
rect_pix.fill(QColor(255,0,0));
scene->addPixmap(rect_pix)->setPos(insert_width+50+80, 2*insert_height+170+30+20+5);

```

```

text = scene->addText(QString("12log(x)"));
text->setPos(insert_width+50+80+10, 2*insert_height+170+30);
text->setFont(textFont);

```

```

text = scene->addText(QString("number of operations(random data)"));
text->setPos(insert_width+50+80+10, 2*insert_height+170+30+20);
text->setFont(textFont);

```

```

QFont axisTitleFont;
axisTitleFont.setPixelSize(10);
axisTitleFont.setBold(true);
axisTitleFont.setFamily("Calibri");

```

```
//1
```

```

text = scene->addText(QString("y = number of operations"));
text->setPos(10,60-5); //y-5
text->setFont(axisTitleFont);

```

```

text = scene->addText(QString("x = data size"));
text->setPos(10+insert_width-80, 60-19+insert_height); //x+wid-80; y-19+height
text->setFont(axisTitleFont);

```

```
//2
```

```

text = scene->addText(QString("y = number of operations"));
text->setPos(10,insert_height+190-5);
text->setFont(axisTitleFont);

```

```

text = scene->addText(QString("x = data size"));

```

```

text->setPos(10+insert_width-80,insert_height+190-19+insert_height);
text->setFont(axisTitleFont);

//3
text = scene->addText(QString("y = number of operations"));
text->setPos(50+insert_width,60-5);
text->setFont(axisTitleFont);

text = scene->addText(QString("x = data size"));
text->setPos(50+insert_width+insert_width-80,60-19+insert_height);
text->setFont(axisTitleFont);

//4
text = scene->addText(QString("y = number of operations"));
text->setPos(50+insert_width,insert_height+190-5);
text->setFont(axisTitleFont);

text = scene->addText(QString("x = data size"));
text->setPos(50+insert_width+insert_width-80,insert_height+190-19+insert_height);
text->setFont(axisTitleFont);
}

QPixmap GraphicWidget::drawLog(QPixmap graph,int width, int height, double x_max_value,
double y_max_value, double log_coeff) {
// QPixmap graph(width, height);
QPainter p(&graph);
p.setRenderHint(QPainter::Antialiasing,true);

QPen pen(QColor(0,0,0));
pen.setWidth(4);
p.setPen(pen);
p.drawLine(0, 0, 0, height-1);
p.drawLine(0, height-1, width-1, height-1);

pen.setColor(QColor(40,77,255));
pen.setWidth(3);
p.setPen(pen);
for(double i=0; i<x_max_value; i+=1) {
    double x = i;
    double y = log_coeff*log(x);
    p.drawPoint(x*width/(x_max_value), (-y*height/y_max_value)+height);
}
return graph;
}

```

```

QPixmap GraphicWidget::createPixmapGraph(int width, int height, double x_max_value, double
y_max_value, double log_coeff) {
    QPixmap graph(width, height);
    QPainter p(&graph);
    graph.fill(QColor(192,192,192));
    p.setRenderHint(QPainter::Antialiasing,true);

    return graph;
}

```

Название файла: GraphicWidget.h

```

#ifndef GRAPHICWIDGET_H
#define GRAPHICWIDGET_H

#include <QObject>
#include <QWidget>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QPixmap>
#include <QPushButton>
#include <QLineEdit>
#include <QVBoxLayout>

class GraphicWidget : public QWidget
{
    Q_OBJECT
public:
    explicit GraphicWidget(QWidget *parent = nullptr);
    void drawStatistics(int data_size=300);
    void clear();
private:
    QPixmap createPixmapGraph(int, int, double, double, double);
    QPixmap drawLog(QPixmap,int,int,double,double,double);
    QGraphicsScene* scene;
    QGraphicsView* view;
    QPushButton* gen_button;
    QVBoxLayout* vbox;
    QLineEdit* gen_input;
    int d_size = 300;
signals:

};

#endif // GRAPHICWIDGET_H

```

Название файла: main.cpp

```
#include "mainwindow.h"
```

```
#include <QApplication>
```

```
#include "AVL_Tree.h"
```

```
#include "treeviewwidget.h"
```

```
#include "graphicwidget.h"
```

```
#include <QRandomGenerator>
```

```
#include <QVector>
```

```
#include <QPushButton>
```

```
#include <QLineEdit>
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    QApplication a(argc, argv);
```

```
    MainWindow w;
```

```
    w.show();
```

```
    return a.exec();
```

```
}
```

Название файла: MainWindow.cpp

```
#include "mainwindow.h"
```

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
```

```
// this->setFixedSize(800,700);
```

```
    this->setFixedSize(900,790);
```

```
    g = new GraphicWidget(this);
```

```
    gen_button = new QPushButton("Generate", this);
```

```
    gen_button->setGeometry(600,740, 100,30);
```

```
    gen_input = new QLineEdit(this);
```

```
    gen_input->setPlaceholderText("Data size (300 by default)");
```

```
    gen_input->setGeometry(200,740, 400,30);
```

```
    connect(gen_button, &QPushButton::clicked, [=]() {
```

```
        QString str = gen_input->text();
```

```
        bool ok;
```

```
        int dec = str.toInt(&ok, 10);
```

```
        if(ok && dec > 0) {
```

```
            g->clear();
```

```
            g->drawStatistics(dec);
```

```
        }
```

```
    });
```

```
}
```

```
MainWindow::~MainWindow() {
```

```
}
```

Название файла: MainWindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "treeviewwidget.h"
#include "graphicwidget.h"
#include <QRandomGenerator>
#include <QVector>
#include <QPushButton>
#include <QLineEdit>
#include <QMainWindow>

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private:
    QPushButton* gen_button;
    QLineEdit* gen_input;
    GraphicWidget* g;
};
#endif // MAINWINDOW_H

```