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: Красно-чёрное дерево

Студент гр. 9303 _____ Халилов Ш.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Халилов Ш.А.

Группа 9303

Тема работы: Красно-чёрные деревя – вставка (Демонстрация)

Исходные данные:

Содержание пояснительной записки:

Аннотация

Введение

Основные теоретические положения.

Описание кода программы

Заключение

Список использованных источников.

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 18.12.2020

Дата защиты реферата: 18.12.2020

Студент _____ Халилов Ш.А.

Преподаватель _____ Филатов Ар.Ю.

АННОТАЦИЯ

Курсовая работа представляет собой программу, предназначенную для демонстрации, связанных с вставкой элементов из красно-чёрное дерева с обеспечением базового функционала задания. Код программы написан на языке программирования C++, запуск программы подразумевается на операционных системах семейства Linux. При разработке кода программы активно использовались функции стандартных библиотек языка C++, основные управляющие конструкции языка C++. Код был написан по парадигме ООП. Для проверки работоспособности программы проводилось тестирование. Исходный код, скриншоты, показывающие корректную работу программы, и результаты тестирования представлены в приложениях.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы — разработка программы для демонстрации добавления элемента в красно-черную древовидную структуру. Также для наиболее удобного взаимодействия с программой был создан графический интерфейс, с использованием Qt.

Для достижения поставленной цели требуется реализовать следующие задачи:

1. Изучение теоретического материала по написанию кода на языке C++ и о работе с красно-черними деревьями.

2. Разработка программного кода в рамках полученного задания.

3. Написание программного кода.

4. Тестирование программного кода.

Полученное задание:

Демонстрация выставки элементов. Тип задания – демонстрации.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Основные теоретические положения о Красно-чёрном дереве.

двоичное дерево поиска, в котором каждый узел имеет атрибут цвета. При этом:

1. Узел может быть либо красным, либо чёрным и имеет двух потомков;

2. Корень — как правило чёрный. Это правило слабо влияет на работоспособность модели, так как цвет корня всегда можно изменить с красного на чёрный;

3. Все листья, не содержащие данных — чёрные.

4. Оба потомка каждого красного узла — чёрные.

5. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов.

Благодаря этим ограничениям, путь от корня до самого дальнего листа не более чем вдвое длиннее, чем до самого ближнего и дерево примерно сбалансировано. Операции вставки, удаления и поиска требуют в худшем случае времени, пропорционального длине дерева, что позволяет красно-чёрным деревьям быть более эффективными в худшем случае, чем обычные двоичные деревья поиска.

Чтобы понять, как это работает, достаточно рассмотреть эффект свойств 4 и 5 вместе. Пусть для красно-чёрного дерева T число чёрных узлов от корня до листа равно B. Тогда кратчайший возможный путь до любого листа содержит B узлов и все они чёрные. Более длинный возможный путь может быть построен путём включения красных узлов. Однако, благодаря п.4 в дереве не может быть двух красных узлов подряд, а согласно пп. 2 и 3, путь начинается и кончается чёрным узлом. Поэтому самый длинный возможный путь состоит из 2H-1 узлов, попеременно красных и чёрных.

Вставка:

Новый узел в красно-чёрное дерево добавляется на место одного из листьев, окрашивается в красный цвет и к нему прикрепляется два листа (так как листья являются абстракцией, не содержащей данных, их добавление не требует дополнительной операции). Что происходит дальше, зависит от цвета близлежащих узлов. Заметим, что:

• Свойство 3 (Все листья чёрные) выполняется всегда.

• Свойство 4 (Оба потомка любого красного узла — чёрные) может нарушиться только при добавлении красного узла, при перекрашивании чёрного узла в красный или при повороте.

• Свойство 5 (Все пути от любого узла до листовых узлов содержат одинаковое число чёрных узлов) может нарушиться только при добавлении чёрного узла, перекрашивании красного узла в чёрный (или наоборот), или при повороте.

Случай 1:

Текущий узел в корне дерева. В этом случае, он перекрашивается в чёрный цвет, чтобы оставить верным Свойство 2 (Корень — чёрный). Так как это действие добавляет один чёрный узел в каждый путь, Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается.

Случай 2:

Предок текущего узла чёрный, то есть Свойство 4 (Оба потомка каждого красного узла — чёрные) не нарушается. В этом случае дерево остаётся корректным. Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается, потому что текущий узел имеет двух чёрных листовых потомков, но так как N является красным, путь до каждого из этих потомков содержит такое же число чёрных узлов, что и путь до чёрного листа, который был заменен текущим узлом, так что свойство остается верным.

Случай 3:

Если и родитель, и дядя — красные, то они оба могут быть перекрашены в чёрный, и дедушка станет красным (для сохранения свойства 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов)). Теперь у текущего красного узла чёрный родитель. Так как любой путь через родителя или дядю должен проходить через дедушку, число чёрных узлов в этих путях не изменится. Однако, дедушка теперь может нарушить свойства 2 (Корень — чёрный) или 4 (Оба потомка каждого красного узла — чёрные) (свойство 4 может быть нарушено, так как родитель может быть красным). Чтобы это исправить, вся процедура рекурсивно выполняется на из случая 1.

Случай 4:

Родитель является красным, но дядя — чёрный. Также, текущий узел — правый потомок, а в свою очередь — левый потомок своего предка. В этом случае может быть произведен поворот дерева, который меняет роли текущего узла и его предка. Тогда, для бывшего родительского узла в обновленной структуре используем случай 5, потому что Свойство 4 (Оба потомка любого красного узла — чёрные) все ещё нарушено. Вращение приводит к тому, что некоторые пути (в поддереве, обозначенном «1» на схеме) проходят через узел, чего не было до этого. Это также приводит к тому, что некоторые пути (в поддереве, обозначенном «3») не проходят через узел. Однако, оба эти узла являются красными, так что Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается при вращении. Однако Свойство 4 всё ещё нарушается, но теперь задача сводится к Случаю 5.

Случай 5:

Родитель является красным, но дядя — чёрный, текущий узел — левый потомок и родитель — левый потомок. В этом случае выполняется поворот дерева на дедушки. В результате получается дерево, в котором бывший родитель теперь является родителем и текущего узла и бывшего дедушки. Известно, что дедушка — чёрный, так как его бывший потомок не мог бы в противном случае быть

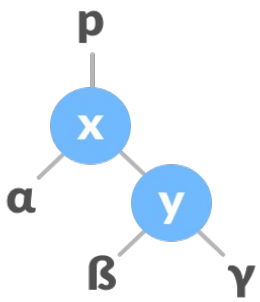
красным (без нарушения Свойства 4). Тогда цвета родитель и дедушка меняются и в результате дерево удовлетворяет Свойству 4 (Оба потомка любого красного узла — чёрные). Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) также остается верным, так как все пути, которые проходят через любой из этих трех узлов, ранее проходили через дедушки, поэтому теперь они все проходят через родителя. В каждом случае, из этих трёх узлов только один окрашен в чёрный.

Повернуть влево:

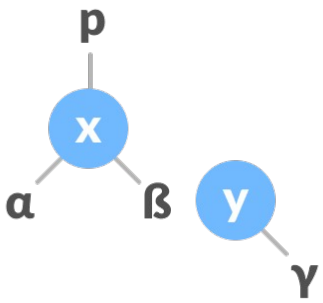
При вращении влево расположение узлов справа преобразуется в расположение узлов слева.

Алгоритм

1. Пусть исходное дерево будет:



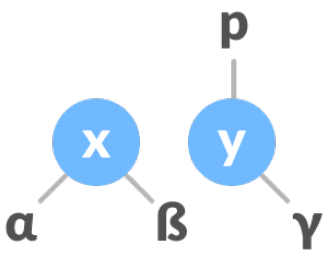
2. Если "y" имеет левое поддерево, назначьте "x" в качестве родителя левого поддерева "y".



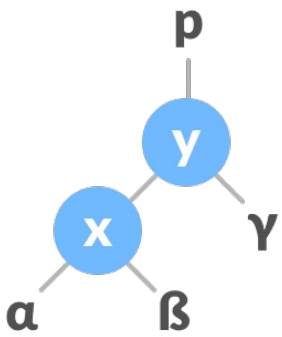
3. Если родитель "x" есть NULL, сделать "y" как корень дерева.

4. Иначе, если "x" левый ребенок "p", сделать "y" как левый ребенок "p".

5. Иначе назначить "y" как правильный ребенок "p"



6. Сделать "y" как родитель "x"

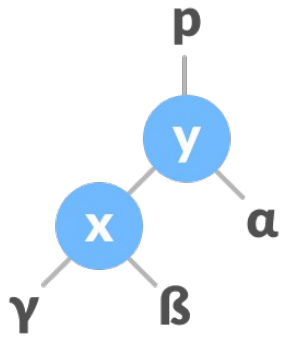


Повернуть влево:

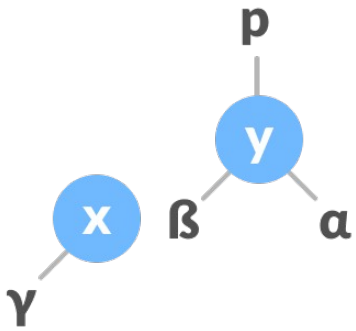
При вращении влево расположение узлов справа преобразуется в расположение узлов слева.

Алгоритм

1. Пусть исходное дерево будет:



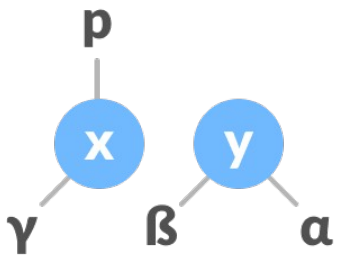
2. Если "x" имеет правое поддерево, назначьте y в качестве родителя правого поддерева "x"



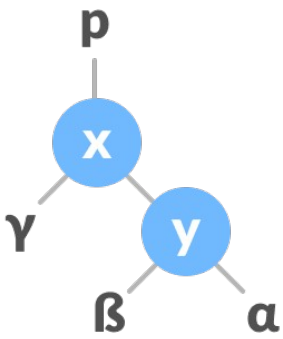
3. Если родитель "y" есть NULL, сделать "x" как корень дерева.

4. Иначе, если "y" правый дочерний элемент своего родителя "p", сделать "x" как правильный ребенок "p"

5. Иначе назначить "x" как левый ребенок "p"



6. Сделать "x" как родитель "y"



1.2. Описание реализованных методов.

Для реализации операций понадобится реализовать две вспомогательные операции: `showing` и `makeGgaph`.

`MakeGgaph()` – это обычный обход дерева, функция создает тело программы на языке dot (язык описания графа), затем этот код генерируется с помощью компилятора (dot) и создаются изображения для демонстрации.

`showing()` – функция обновления изображения на графическом экране эта функция вызывается через определенное время, которое пользователь может контролировать

`Update()` – Метод, создающий картинку, визуализирующую текущее построенное дерево. Создает текстовый файл, описывающий структуру дерева, а также .png изображение, используя dot компилятор.

2. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

2.1. Описание главного окна программы

Для осуществления взаимодействия с пользователем в ходе курсовой работы был реализован графический интерфейс с использованием фреймворка `Qt`. Ядром интерфейса является главное окно, для описания которого был реализован класс `MainWindow`.

Класс `MainWindow` имеет следующие поля:.

`QGraphicsScene* scene` – сцена, на которой отображается задание (картинка красно-чёрное дерева).

Была написана структура `struct Item`, которая имеет следующие поля, это: path поля для хранения пути к изображению на соответствующем шаге и comment поля для комментариев к изображению.

Также класс `MainWindow` содержит следующие методы:

`generation()` – функция генерации элементов для дерева, она генерирует определенное количество раз случайное число и вызывает `insertElem` для добавления элемента

`insertElem()` – функция добавления элемента в структуру, она проверяет наличие элемента в структуре и если такого элемента нет, то передает его на добавление, в противном случае элемент не добавляется, так как из-за особенностей языка (dot) в структуре невозможно найти два одинаковых элемента

`clean()` - функция очистки памяти и сцены.

`Add()` - функция добавления элемента в структуру, но в этой функции элемент добавляется по одному элементу и входное значение вводится пользователем

`documentation()` – функция отображения информации о красно-черном дереве.

Основные функции:

InsertRecurse() - функция ищет место для элемента и вставляет элемент в это место

Insert()- функция получает элемент и выдает его для добавления в лес, после выдает элемент для проверки свойства

fixProperties() - функция для исправленных свойств красно-черного дерева, после вставки

RotateLeft() - функция поворота дерева влево

RotateRight() - функция поворота дерева вправо

findElem() - функция поиска элемента в дереве

CountElem() - функция для подсчета количества вхождений элемента в дерево

free() - функция очистки памяти под элементом

2.2. Описание пользовательского интерфейса.

главное окно разделено на две части, слева находится сцена, на которой отображается изображение дерева, и обновляется на каждом шаге процесса добавления элементов, а правая часть окна также разделена на две: верхняя отображает комментарии, а нижняя - компоненты для управления. Исходный код программы представлен в приложении Б. Примеры работы программы - в приложении А.

ЗАКЛЮЧЕНИЕ

Для успешного достижения поставленной цели — написания программы для демонстрации просцесс виставки элемента в структуру как красно-черное дерево, соответствующих заданию курсовой работы, были выполнены соответствующие задачи:

1. Изучен теоретический материал по теме курсовой работы.
2. Разработан программный код.
3. Реализован программный код.
4. Проведено тестирование программы.

Исходный код программы представлен в приложении Б, Примеры работы программы - в приложении А.

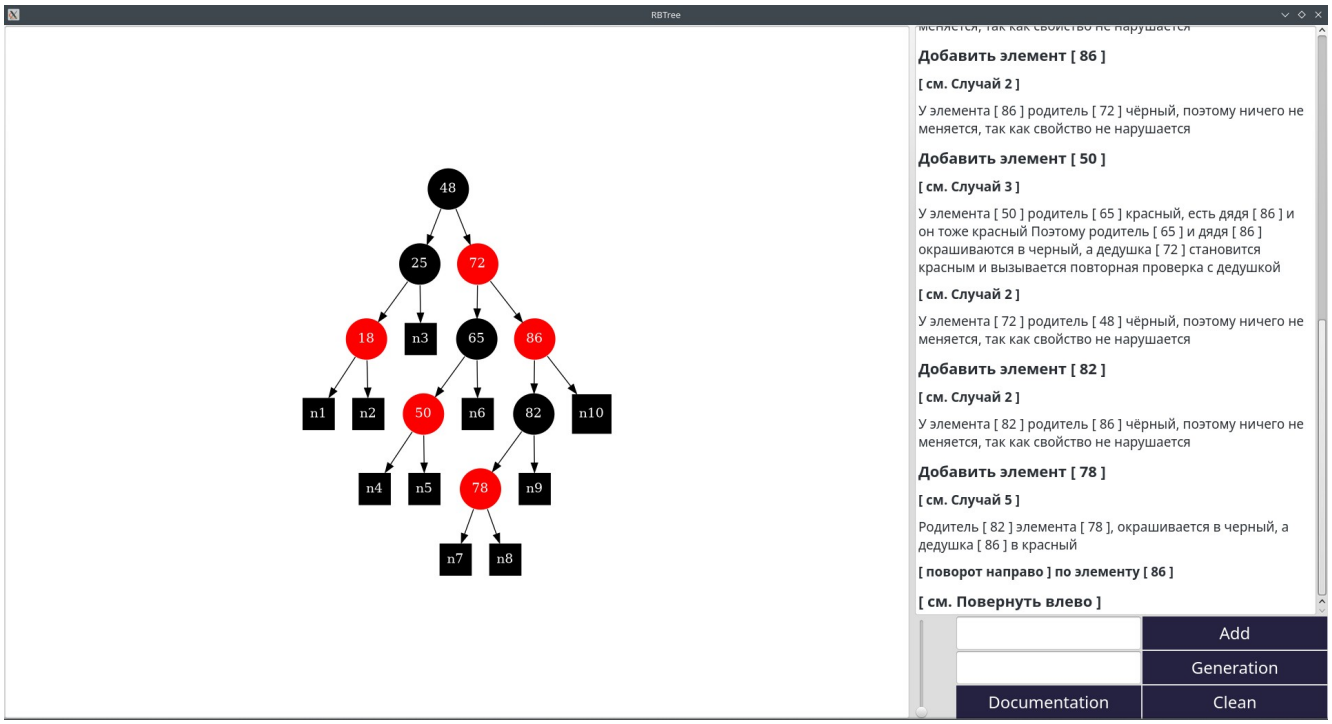
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

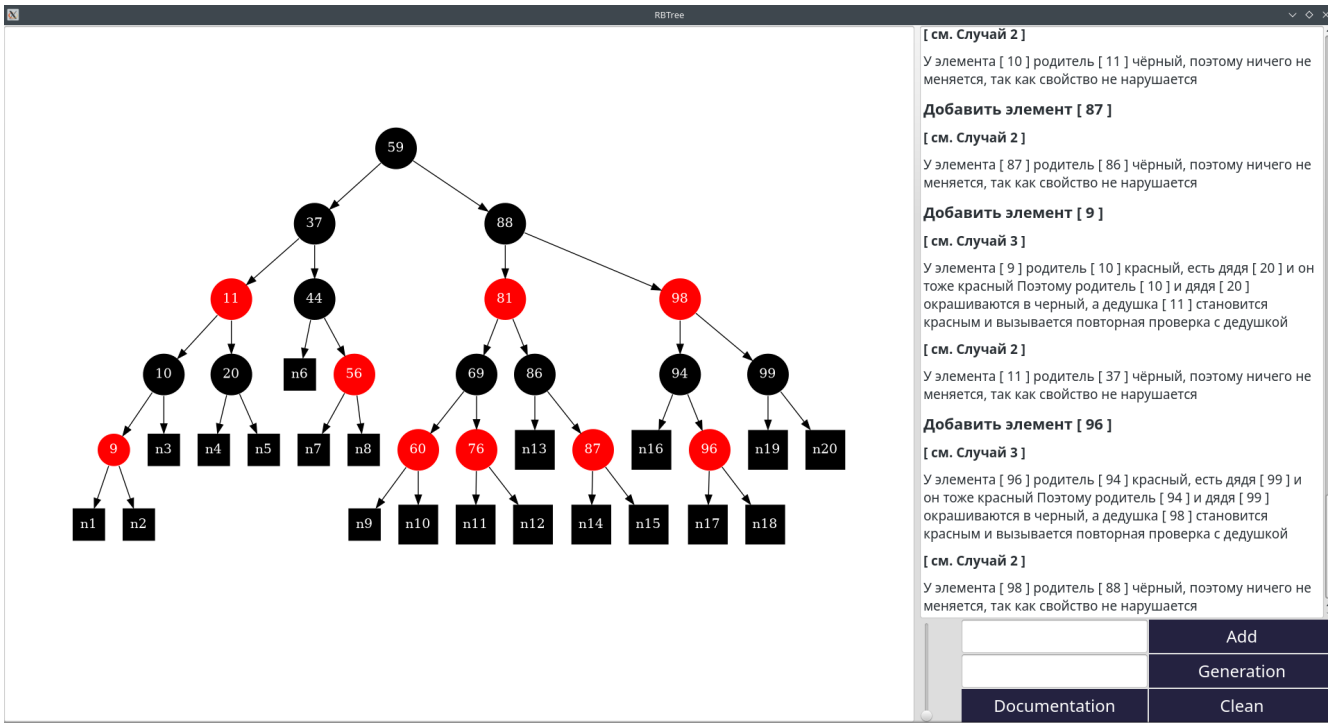
1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с.

2. Основы программирования на языках Си и C++ [Электронный ресурс URL: http://cplusplus.com

ПРИЛОЖЕНИЕ А

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ





ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

MAINWINDOW.H:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include "documentation.h"
#include "helperfunctions.h"


class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    QTimer Tm;
    int interval = 1000;
    void InsertRecurse(Node *current, Node *elem);
    void RotateLeft(Node *elem);
    void RotateRight(Node *elem);
    void fixProperties(Node *elem);
    void Insert(Node *&root, Node *elem);
    void makeGgaph(Node *elem, std::string  &str, int &n);
    std::string comments ="";
    std::list<Item*> items;
    QTextBrowser *TextEdit;
    QSlider *slider = nullptr;
public:
    QPushButton *ButtonAdd = nullptr;
    QPushButton *ButtonDoc = nullptr;
    QPushButton *genration = nullptr;
    QPushButton *Clean = nullptr;
    QLineEdit *lineEdit = nullptr;
    QLineEdit *lineEditTwo = nullptr;
    int count = 0;
    int counter = 0;
```

```
    int len = 0;
    int *Arr = nullptr;
    Node *Head = nullptr;
    MainWindow(QWidget *parent = nullptr);

    QGraphicsScene *Scene;
    QGraphicsView *View;
    QWidget* CentralWidget;
    ~MainWindow();
    void update(std::string str);
    void insertElem(int x);
    void setScene();
    void showing();
public slots:
    void clean();
    void generation();
    void add();
    void documentation();

};
#endif // MAINWINDOW_H
```

NODE.HPP:

```
#ifndef NODE_HPP

#define NODE_HPP

#include "libs.h"

#define DIGRAPH "digraph G {\n"

#define ENDGRAPH "\n}"

#define BLACKTREE  " [ shape = circle, style=filled,
fillcolor=black];\n"
```

```
#define REDTREE " [ shape = circle, style=filled, fillcolor=red];\n"

#define CURRENTTREE " [ shape = doublecircle, style=filled, color =
blue, fillcolor=red];\n"

#define NILL " [shape = square, style=filled, fillcolor=black ];\n"

#define NODESTYLE "node[ color = white, fontcolor=white];\n";

#define RED Qt::red

#define BLACK Qt::black


struct Item {


    std::string path;

    std::string comment;

    Item(std::string p, std::string c):path(p), comment(c){}

};


class Node : public QGraphicsEllipseItem

{

private:

    int value;

    QColor color = RED;

    Node *parent;
```

```
    Node *left;
    Node *right;
public:
    Node(int val, QColor cl): value(val), color(cl){
        parent = left = right = nullptr;
        setRect(0, 0, 50, 50);
    }
    Node *getParent() const;
    void setParent(Node *value);
    Node *getLeft() const;
    void setLeft(Node *value);
    Node *getRight() const;
    void setRight(Node *value);


    QColor getColor() const;
    void setColor(const QColor &value);
    int getValue() const;
    void setValue(int value);
    ~Node(){
        if( left) delete left;
        if( right ) delete right;
```

```
    }
};


#endif // NODE_HPP
```

DOCUMENTATION.H

```
#ifndef DOCUMENTATION_H
#define DOCUMENTATION_H

#include "libs.h"

class Documentation : public QDialog
{
    Q_OBJECT
private:
    QPushButton *close;
    QGridLayout *grid_layout;
    QScrollArea *ScrolArea;
    QTextBrowser *info;

public:
    Documentation(QWidget *parent = 0);
    ~Documentation();
};

#endif // DOCUMENTATION_H
```

HELPERFUNCTIONS.H:

```
#ifndef HELPERFUNCTIONS_H
#define HELPERFUNCTIONS_H
#include "node.hpp"

void createForest(Node *&root, int *arr, int size);
bool isRed(Node *elem);
bool isBlack(Node *elem);
Node *Parent(Node *elem);
Node *GrandParent(Node *elem);
bool isLeft(Node *elem);
bool isRight(Node *elem);
Node *Brother(Node *elem);
Node *Uncle(Node *elem);
Node * Root(Node* elem);;
Node *findElem(Node *root, int key);
void free(Node *elem);
bool isLeaf(Node *elem);
bool isNotLeaf(Node *elem);
int CountElem(Node *elem, int key);
#endif // HELPERFUNCTIONS_H
```

MAIN.CPP:

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

MAINWINDOW.CPP:

```
#include "mainwindow.h"

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

    CentralWidget = new QWidget();
    setScene();
    setMinimumSize(800, 600);
    setCentralWidget(CentralWidget);

    ButtonAdd = new QPushButton("Add");
    ButtonDoc = new QPushButton("Documentation");
    genration = new QPushButton("Generation");
    Clean = new QPushButton("Clean");
    TextEdit = new QTextBrowser();
    TextEdit->setReadOnly(true);

    lineEdit = new QLineEdit();
    lineEditTwo = new QLineEdit();
    lineEdit->setValidator(new QRegExpValidator(QRegExp("^[0-9]
{0,4}"), lineEdit));
    lineEditTwo->setValidator(new QRegExpValidator(QRegExp("^[0-9]
{0,2}"), lineEditTwo));

    slider = new QSlider(this);
    slider->setTickInterval(100);
    slider->setMaximum(5000);
    slider->setMinimum(1000);
    slider->setFixedSize(60, 150);

    QHBoxLayout *main_Layout = new QHBoxLayout();
    QVBoxLayout *VBLayout = new QVBoxLayout();
    QHBoxLayout *HBLayout = new QHBoxLayout();
    QHBoxLayout *HBLayout2 = new QHBoxLayout();
    QHBoxLayout *HBLayout3 = new QHBoxLayout();

    QVBoxLayout *VBLayout2 = new QVBoxLayout();
    QHBoxLayout *HBLayout4 = new QHBoxLayout();

    ButtonAdd->setFixedSize(270, 50);
    ButtonDoc->setFixedSize(270, 50);
    genration->setFixedSize(270, 50);
    Clean->setFixedSize(270, 50);
    lineEdit->setFixedSize(270, 50);
    lineEditTwo->setFixedSize(270, 50);
    TextEdit->setFixedWidth(600);

    VBLayout2->addItem(HBLayout);
    VBLayout2->addItem(HBLayout2);
```

```
    VBLayout2->addItem(HBLayout3);
    HBLayout4->addWidget(slider);
    HBLayout4->addItem(VBLayout2);

    HBLayout->addWidget(lineEdit);
    HBLayout->addWidget(ButtonAdd);

    HBLayout2->addWidget(lineEditTwo);
    HBLayout2->addWidget(genration);

    HBLayout3->addWidget(ButtonDoc);
    HBLayout3->addWidget(Clean);

    VBLayout->addWidget(TextEdit);
    VBLayout->addItem(HBLayout4);

    HBLayout2->setContentsMargins(0, 0, 0,0);
    HBLayout->setContentsMargins(0, 0, 0,0);
    main_Layout->setContentsMargins(0, 0, 0,0);
    VBLayout->setContentsMargins(0, 0, 0,0);
    HBLayout4->setContentsMargins(0, 0, 0,0);
    VBLayout2->setContentsMargins(0, 0, 0,0);

    main_Layout->addWidget(View);
    main_Layout->addItem(VBLayout);

    showMaximized();
    connect(ButtonAdd, &QPushButton::clicked, this,
&MainWindow::add);
    connect(Clean, &QPushButton::clicked, this, &MainWindow::clean);
    connect(genration, &QPushButton::clicked, this,
&MainWindow::generation);
    connect(ButtonDoc, &QPushButton::clicked, this,
&MainWindow::documentation);
    connect(slider, &QSlider::valueChanged, this, [&](){
        interval = slider->value();
        Tm.setInterval(interval);
    });
    View->show();
    Tm.setInterval(interval);
    connect(&Tm, &QTimer::timeout, this, [&](){
       showing();
    });
    CentralWidget->setLayout(main_Layout);
    setStyleSheet("QPushButton{                                    "
                "   background-color: #242240;                     "
                "   color: white;                                  "
                "   height: 50px;                                  "
                "   width: 300px;                                  "
                "   margin: 1px 1px;                               "
                "   border-radius: 0;                              "
```

```
                "   border: 2px;                                                "
                "   font-size: 18pt;                                            "
                "}                                                              "
                "QPushButton:hover {                                            "
                "   background-color: rgb(255, 69, 0);                          "
                "}                      "
                "QTextBrowser{"
                   "font-size: 14pt;"
                   ""
                "}"
                   ".QSlider::groove:horizontal {"
                   "border: 1px solid #999999;"
                   "height: 8px;"
                   "background: qlineargradient(x1:0, y1:0, x2:0,
y2:1, stop:0 #B1B1B1, stop:1 #c4c4c4);"
                   "margin: 2px 0;"
            "  }"
            ".QSlider::handle:horizontal {"
"                   background: qlineargradient(x1:0, y1:0, x2:1,
y2:1, stop:0 #b4b4b4, stop:1 #8f8f8f);"
"                   border: 1px solid #5c5c5c;"
"                  width: 18px;"
"                   margin: -2px 0; "
             "   border-radius: 3px;"
             "}");

}

void MainWindow::documentation() {

    Documentation doc;
    doc.exec();
}

MainWindow::~MainWindow()
{
    if( Head ) delete Head;
    system("rm *.png");
    system("rm *.dot");
}

void MainWindow::update(std::string str)
{
    int nn=0;
    std::string grap = "";
    makeGgaph(Head, grap, nn);
    std::ofstream graphic("tree.dot");
    graphic << DIGRAPH;
    graphic << NODESTYLE;
    graphic <<grap;
    graphic << ENDGRAPH;
```

```
    graphic.close();

    std::string fileName = "tree"+std::to_string(count++)+".png";
    std::string sysComand = "dot tree.dot -o "+fileName+" -Tpng";
    system(sysComand.c_str());

    items.push_back(new Item( "./"+fileName, str));
}

void MainWindow::insertElem(int x){
    if(findElem(Head, x)){
        update("<p>В структуре этот элемент [ "+ std::to_string(x) +
" ] уже существует</p>\n");
        return;
    }
    update("<h3>Добавить элемент [ "+std::to_string(x)+" ]</h3>\n");
    Insert(Head, new Node( x, RED));
}

void MainWindow::showing(){

    if(counter == count ){

        Tm.stop();
        return;
    }

    std::list<Item*>::iterator it = items.begin() ;

    std::advance(it, counter++);
    QImage image(QString::fromStdString((*it)->path));
    QGraphicsPixmapItem* item = new
QGraphicsPixmapItem(QPixmap::fromImage(image));
    comments += (*it)->comment +"\n";
    Scene->clear();
    Scene->addItem(item);
    View->update();

    TextEdit->setHtml(QString::fromStdString(comments));
    TextEdit->verticalScrollBar()->setValue(TextEdit-
>verticalScrollBar()->maximum());
}

void MainWindow::clean()
{
    if( Head ) delete Head;
    Head = nullptr;
    comments = "";
    TextEdit->setText(QString::fromStdString(comments));
    count = 0;
    counter = 0;
```

```
    items.clear();
    Scene->clear();
    system("rm *.png");
    system("rm *.dot");
}

void MainWindow::generation()
{
    int len;
    if(lineEditTwo->text().isEmpty()){
        len = 10;
    }else {
        len = lineEditTwo->text().toInt();
    }
    lineEditTwo->clear();

    srand(time(0));
    for (int i = 0; i < len ; i++ ) {
         insertElem(rand()%100);
    }
    Tm.start();
}

void MainWindow::add()
{
    if(lineEdit->text().isEmpty())
        return;

    int tmp = lineEdit->text().toInt();
    lineEdit->clear();
    if(findElem(Head, tmp)){
        update("<p>В структуре этот элемент [ "+ std::to_string(tmp)
+ " ] уже существует</p>\n");
        return;
    }
    Insert(Head, new Node(tmp, RED));
    update("<h3>Добавить элемент [ "+std::to_string(tmp)+" ]</h3>\
n");
    Tm.start();
}

void MainWindow::setScene()
{
    Scene = new QGraphicsScene(CentralWidget);
    Scene->setBackgroundBrush(Qt::white);
    View = new QGraphicsView(Scene);
}

void MainWindow::InsertRecurse(Node *current, Node *elem)
{
```

```
    //? Рекурсивно спускайтесь по дереву до тех пор, пока не будет
найден лист.
    if (current != nullptr)
    {
        //? если текущее значение больше чем новое значение, то
слева от этого
        if (current->getValue() > elem->getValue())
        {
            //? если левая сторона узла не пуста
            if (current->getLeft() != nullptr)
            {
                //* Рекурсивно спускаться влево-вниз
                InsertRecurse(current->getLeft(), elem);

                return;
            }

            //? если левая сторона узла пуста
            //* присваивать значение на левом узле
            current->setLeft(elem);

        }
        else //? если текущее значение меньше чем новое значение, то
справа от этого
        {
            //? если правая сторона узла не пуста
            if (current->getRight() != nullptr)
            {
                //* Рекурсивно спускаться право-вниз
                InsertRecurse(current->getRight(), elem);

                return;
            }

            //? если правая сторона узла пуста
            //* присваивать значение на правом узле
            current->setRight(elem);

        }
    }
    //* сделать текущий элемент родительским для нового элемента
    elem->setParent(current);

}

void MainWindow::fixProperties(Node *elem)
{
    std::string comment ="";

    if (!Parent(elem))
    {
```

```
        comment = "<h4>[ см. Случай 1 ]</h4>";
        comment +="<p>Элемент [ "+ std::to_string(elem->getValue())
+ " ] стал корнем дерева</p>";
        update(comment);
        elem->setColor(BLACK);
        return;
    }
    else if (isBlack(Parent(elem)))
    {
        comment = "<h4>[ см. Случай 2 ]</h4>";
        comment +="<p>У элемента [ "+ std::to_string(elem-
>getValue()) + " ] родитель [ "+std::to_string(Parent(elem)-
>getValue()) +" ] чёрный, "
       "поэтому ничего не меняется, так как свойство не нарушается</
p>";
        update(comment);
        return;
    }
    else if (Uncle(elem) != nullptr && isRed(Uncle(elem)))
    {
        comment = "<h4>[ см. Случай 3 ]</h4>";
        comment += "<p>У элемента [ "+ std::to_string(elem-
>getValue()) + " ] родитель [ "+std::to_string(Parent(elem)-
>getValue()) +" ] красный, есть дядя [ "+std::to_string(Uncle(elem)-
>getValue()) +" ] и он тоже красный "
"Поэтому родитель [ "+std::to_string(Parent(elem)->getValue()) +" ]
и дядя [ "+std::to_string(Uncle(elem)->getValue()) +" ] окрашиваются
в черный, а дедушка [ "+std::to_string(GrandParent(elem)-
>getValue()) +" ] "
"становится красным и вызывается повторная проверка с дедушкой</p>";

        update(comment);

        Parent(elem)->setColor(BLACK);
        Uncle(elem)->setColor(BLACK);
        GrandParent(elem)->setColor(RED);
        fixProperties(GrandParent(elem));

        return;
    }
    else if (isRight(elem) && isLeft(Parent(elem)))
    {
        comment = "<h4>[ см. Случай 4 ]</h4>";
        comment += "<p>Элемент [ "+ std::to_string(elem->getValue())
+ " ] является левым, а родитель [ "+std::to_string(Parent(elem)-
>getValue()) +" ] является\n"
"правым элементом, поэтому делаем поворот направо вокруг родителя
[ "+std::to_string(Parent(elem)->getValue()) +" ]</p>";

        update(comment);
```

```
        RotateLeft(Parent(elem));
        elem = elem->getLeft();

    }
    else if (isLeft(elem) && isRight(Parent(elem)))
    {
        comment = "<h4>[ см. Случай 4 ]</h4>";
        comment +="<p>Элемент [ "+ std::to_string(elem->getValue())
+ " ] является правым, а родитель [ "+std::to_string(Parent(elem)-
>getValue()) +" ] является "
"левым элементом, поэтому делаться поворот налево вокруг родителя
[ "+std::to_string(Parent(elem)->getValue()) +" ]</p>";

update(comment);
        RotateRight(Parent(elem));
        elem = elem->getRight();
    }

    comment = "<h4>[ см. Случай 5 ]</h4>";
    comment +="<p>Родитель [ "+std::to_string(Parent(elem)-
>getValue()) +" ] элемента [ "+std::to_string(elem->getValue()) +
" ], окрашивается в черный, а дедушка
[ "+std::to_string(GrandParent(elem)->getValue()) +" ] в
красный</p>";

    update(comment);
    Parent(elem)->setColor(BLACK);
    GrandParent(elem)->setColor(RED);

    if (isLeft(elem) && isLeft(Parent(elem)))
    {
        comment ="<p>Элемент [ "+ std::to_string(elem->getValue()) +
" ] является левым узлом. Родитель [ "+std::to_string(elem-
>getValue()) + " ] также является левым узлом, поэтому делаться
поворот направо вокруг дедушки [ "+std::to_string(GrandParent(elem)-
>getValue()) +" ]</p>";

        RotateRight(GrandParent(elem));
    }
    else
    {
        comment ="<p>Элемент [ "+ std::to_string(elem->getValue()) +
" ] является правым узлом. Родитель [ "+std::to_string(elem-
>getValue()) + " ] также является правым узлом, поэтому делаться
поворот налево вокруг дедушки [ "+std::to_string(GrandParent(elem)-
>getValue()) +" ]</p>";

        RotateLeft(GrandParent(elem));
    }
}
```

```
void MainWindow::Insert(Node *&root, Node *elem)
{
    //* Вставить новый узел в текущее дерево.
    InsertRecurse(root, elem);

    //* Восстановите дерево в случае нарушения какого-либо из
красно-черных свойств.
    fixProperties(elem);
    //* Найти новый корень для возврата.
    root = elem;
    while (root->getParent())
    {
        root = root->getParent();
    }
}

void MainWindow::makeGgaph(Node *elem, std::string &str, int &n){

    if( elem == nullptr ){
         str += "n"+std::to_string(n)+NILL;
         return;
    }
    if(isRed(elem)){
        str += std::to_string(elem->getValue())+REDTREE;
    }
    else {
        str += std::to_string(elem->getValue())+BLACKTREE;
    }

    if(isNotLeaf(elem->getLeft())){

        str += std::to_string(elem->getValue())+" ->
"+std::to_string(elem->getLeft()->getValue())+"\n";
        makeGgaph(elem->getLeft(), str, n);
    }
    else{
        n++;
        str += std::to_string(elem->getValue())+" ->
n"+std::to_string(n)+"\n";
        str += "n"+std::to_string(n)+NILL;
    }

    if( isNotLeaf(elem->getRight()) ){
       str += std::to_string(elem->getValue())+" ->
"+std::to_string(elem->getRight()->getValue())+"\n";
        makeGgaph(elem->getRight(), str, n);
    }
    else{
        n++;
        str += std::to_string(elem->getValue())+" ->
n"+std::to_string(n)+"\n";
```

```
        str +="n"+ std::to_string(n)+NILL;
    }
}

void MainWindow::RotateLeft(Node *elem)
{
    std::string comment = "<h4>[ поворот налево ] по элементу
[ "+std::to_string( elem->getValue()) +" ]</h4>"

"<h3>[ см. Повернуть влево ]</h3>";
    update(comment);
    Node *Son = elem->getRight();
    Node *parent = Parent(elem );
    Son->setParent( parent ); //* при этом, возможно, Son становится
корнем дерева

    if ( parent )
    {
        if (isLeft(elem))
        {
            parent->setLeft( Son );
        }
        else
        {
            parent->setRight( Son );
        }
    }

    elem->setRight( Son->getLeft() );
    if (Son->getLeft())
    {
        Son->getLeft()->setParent( elem );
    }
    elem->setParent( Son );
    Son->setLeft( elem );
}

void MainWindow::RotateRight(Node *elem)
{

    std::string comment = "<h4>[ поворот направо ] по элементу
[ "+std::to_string( elem->getValue()) +" ]</h4>"

"<h3>[ см. Повернуть влево ]</h3>";;
    Node *Son = elem->getLeft();
    Node *parent = Parent(elem );
    update(comment);
    Son->setParent( parent ); //* при этом, возможно, Son становится
корнем дерева

    if (parent)
```

```
    {
        if (isLeft(elem))
        {
            parent->setLeft( Son );
        }
        else
        {
            parent->setRight( Son );
        }
    }
    elem->setLeft( Son->getRight() );
    if ( Son->getRight() ){
        Son->getRight()->setParent( elem );
    }
    elem->setParent( Son );
    Son->setRight( elem );
}
```

NODE.CPP

```
#include "node.hpp"

Node *Node::getParent() const
{
    return parent;
}

void Node::setParent(Node *value)
{
    parent = value;
}

Node *Node::getLeft() const
{
    return left;
}

void Node::setLeft(Node *value)
{
    left = value;
```

```
}

Node *Node::getRight() const
{
    return right;
}

void Node::setRight(Node *value)
{
    right = value;
}

QColor Node::getColor() const
{
    return color;
}

void Node::setColor(const QColor &value)
{
    this->setBrush(value);
    color = value;
}

int Node::getValue() const
{
    return value;
}

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

DOCUMENTATION.CPP:

```
#include "documentation.h"
```

```
Documentation::Documentation(QWidget *parent) :QDialog(parent) {

    QString Documentation;
    close = new QPushButton("&close", this);
    grid_layout = new QGridLayout();
    ScrolArea = new QScrollArea(this);

    info = new QTextBrowser(ScrolArea);
    info->setText(Documentation);
    grid_layout->addWidget(info, 0, 0, 1, 1);
    grid_layout->addWidget(close, 1, 0, 1, 1, Qt::AlignRight);
    setLayout(grid_layout);
    info->setAlignment(Qt::AlignCenter);
    QString style_buttom = "QPushButton{ "
                           "background-color: #00bdaa;"
                           "height: 40px;"
                           "width: 100px;"
                           "margin:0;"
                           "border-radius: 0;"
                       "}"
                       "QPushButton:hover { "
                           "background-color: rgb(255, 69, 0);"
                        "}";

    close->setStyleSheet(style_buttom);
    setFixedSize(800, 800);
    connect(close, SIGNAL(clicked()), this, SLOT(close()));
    setWindowTitle("Documentation");
    setLayout(grid_layout);
    setStyleSheet("QTextBrowser{"
                  "font-size: 14pt;"
                  "}");
    info->setHtml(
"                <h1>Красно-чёрное дерево</h1>"
"                <p>двоичное дерево поиска, в котором каждый узел
имеет атрибут цвета. При этом:</p>"
"                <ol>"
"                <li>"
"                  Узел может быть либо красным, либо чёрным и имеет
двух потомков;  "
"                </li>"
"                <li>"
"                  Корень — как правило чёрный. Это правило слабо
влияет на работоспособность модели, так как цвет корня всегда можно
изменить с красного на чёрный;"
"                </li>"
"                <li>Все листья, не содержащие данных —
чёрные.</li>"
"                <li>"
"                  Оба потомка каждого красного узла — чёрные."
"                </li>"
```

```
"                <li>"
"                Любой простой путь от узла-предка до листового
узла-потомка содержит одинаковое число чёрных узлов.  "
"                </li>"
"                </ol>"
"                <p>"
"                Благодаря этим ограничениям, путь от корня до
самого дальнего листа не более чем вдвое длиннее, чем до самого
ближнего и дерево примерно сбалансировано. Операции вставки,
удаления и поиска требуют в худшем случае времени, пропорционального
длине дерева, что позволяет красно-чёрным деревьям быть более
эффективными в худшем случае, чем обычные двоичные деревья поиска.</
p>"
"                <p>Чтобы понять, как это работает, достаточно
рассмотреть эффект свойств 4 и 5 вместе. Пусть для красно-чёрного
дерева T число чёрных узлов от корня до листа равно B. Тогда
кратчайший возможный путь до любого листа содержит B узлов и все они
чёрные. Более длинный возможный путь может быть построен путём
включения красных узлов. Однако, благодаря п.4 в дереве не может
быть двух красных узлов подряд, а согласно пп. 2 и 3, путь
начинается и кончается чёрным узлом. Поэтому самый длинный возможный
путь состоит из 2H-1 узлов, попеременно красных и чёрных.</p>"
"                </p>"
"                <h1> Вставка </h1>"
"                <p > Новый узел в красно-чёрное дерево добавляется
на место одного из листьев, окрашивается в красный цвет и к нему
прикрепляется два листа (так как листья являются абстракцией, не
содержащей данных, их добавление не требует дополнительной
операции). Что происходит дальше, зависит от цвета близлежащих
узлов. Заметим, что:</p>"
"                <ul>"
"                <li><b>Свойство 3</b> (Все листья чёрные)
выполняется всегда.</li>"
"                <li><b>Свойство 4</b> (Оба потомка любого красного
узла — чёрные) может нарушиться только при добавлении красного узла,
при перекрашивании чёрного узла в красный или при повороте.</li>"
"                <li><b>Свойство 5</b> (Все пути от любого узла до
листовых узлов содержат одинаковое число чёрных узлов) может
нарушиться только при добавлении чёрного узла, перекрашивании
красного узла в чёрный (или наоборот), или при повороте.</li>"
"                </ul>"
"                <h3> Случай 1:</h3>"
"                <p> Текущий узел в корне дерева. В этом случае, он
перекрашивается в чёрный цвет, чтобы оставить верным Свойство 2
(Корень — чёрный). Так как это действие добавляет один чёрный узел в
каждый путь, Свойство 5 (Все пути от любого данного узла до листовых
узлов содержат одинаковое число чёрных узлов) не нарушается."
"                </p>"
"                <h3> Случай 2:</h3>"
"                <p>"
"                Предок текущего узла чёрный, то есть Свойство 4
(Оба потомка каждого красного узла — чёрные) не нарушается. В этом
```

случае дерево остаётся корректным. Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается, потому что текущий узел имеет двух чёрных листовых потомков, но так как N является красным, путь до каждого из этих потомков содержит такое же число чёрных узлов, что и путь до чёрного листа, который был заменен текущим узлом, так что свойство остается верным."
"   "
"   <h3> Случай 3:</h3>"
"   "
"   Если и родитель, и дядя — красные, то они оба могут быть перекрашены в чёрный, и дедушка станет красным (для сохранения свойства 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов)). Теперь у текущего красного узла чёрный родитель. Так как любой путь через родителя или дядю должен проходить через дедушку, число чёрных узлов в этих путях не изменится. Однако, дедушка теперь может нарушить свойства 2 (Корень — чёрный) или 4 (Оба потомка каждого красного узла — чёрные) (свойство 4 может быть нарушено, так как родитель может быть красным). Чтобы это исправить, вся процедура рекурсивно выполняется на из случая 1."
"   "

"   <h3> Случай 4:</h3>"
               " "
               "Родитель является красным, но дядя — чёрный. Также, текущий узел — правый потомок, а в свою очередь — левый потомок своего предка. В этом случае может быть произведен поворот дерева, который меняет роли текущего узла и его предка. Тогда, для бывшего родительского узла в обновленной структуре используем случай 5, потому что Свойство 4 (Оба потомка любого красного узла — чёрные) все ещё нарушено. Вращение приводит к тому, что некоторые пути (в поддереве, обозначенном «1» на схеме) проходят через узел, чего не было до этого. Это также приводит к тому, что некоторые пути (в поддереве, обозначенном «3») не проходят через узел. Однако, оба эти узла являются красными, так что Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается при вращении. Однако Свойство 4 всё ещё нарушается, но теперь задача сводится к Случаю 5."
               " "
               "<h3> Случай 5:</h3>"
               " "
"   Родитель является красным, но дядя — чёрный, текущий узел — левый потомок и родитель — левый потомок. В этом случае выполняется поворот дерева на дедушки. В результате получается дерево, в котором бывший родитель теперь является родителем и текущего узла и бывшего дедушки. Известно, что дедушка — чёрный, так как его бывший потомок не мог бы в противном случае быть красным (без нарушения Свойства 4). Тогда цвета родитель и дедушка меняются и в результате дерево удовлетворяет Свойству 4 (Оба потомка любого красного узла — чёрные). Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных

```
узлов) также остается верным, так как все пути, которые проходят
через любой из этих трех узлов, ранее проходили через дедушки,
поэтому теперь они все проходят через родителя. В каждом случае, из
этих трёх узлов только один окрашен в чёрный."
"                    </p>"
    "<h3>Повернуть влево</h3>"
    "<p>При вращении влево расположение узлов справа преобразуется в
расположение узлов слева.</p>"
    "<h4>Алгоритм</h4>"
    "<ol>"
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Пусть исходное дерево будет:"
    "       </font>"
            "<br>"
    "       <figure>"
    "          <img src =\":/img/res/tree_l1.png\">"
    "       </figure>"
    "   </li>"
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Если у имеет левое поддерево, назначьте \"x\" в
качестве родителя левого поддерева \"y\"."
    "       </font>"
    "       <br>"
    "       <figure>"
    "           <img src =\":/img/res/tree_l2.png\">"
    "       </figure>"
    "   </li>"
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Если родитель \"x\" есть NULL, сделатьу как корень
дерева."
    "       </font>"
    "       <br>"
    "   </li>"
    "   <li>"
    "      <font style=\"vertical-align: inherit;\">"
    "           Иначе, если \"x\" левый ребенок \"p\", сделать \"y\"
как левый ребенок \"p\"."
    "      </font>"
    "       <br>"
    "   </li>"
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Иначе назначить \"y\" как правильный ребенок \"p\""
    "       </font>"
    "       <br>"
    "       <figure>"
    "          <img src =\":/img/res/tree_l3.png\">"
    "       </figure>"
    "   </li>"
```

```
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Сделать \"y\" как родитель \"x\""
    "       </font>"
    "       <br>"
    "       <figure>"
    "          <img src =\":/img/res/tree_l4.png\">"
    "       </figure>"
    "   </li>"
    "</ol>"
    ""
    "<h3>Повернуть влево</h3>"
    "<p>При вращении влево расположение узлов справа преобразуется в
расположение узлов слева.</p>"
    "<h4>Алгоритм</h4>"
    "<ol>"
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Пусть исходное дерево будет:"
    "       </font>"
    "       <br>"
    "       <figure>"
    "          <img src =\":/img/res/tree_r1.png\">"
    "       </figure>"
    "   </li>"
    "   <li>"
    "      <font style=\"vertical-align: inherit;\">"
    "         Если \"x\" имеет правое поддерево, назначьте y в
качестве родителя правого поддерева \"x\""
    "      </font>"
    "       <br>"
    "      <figure>"
    "          <img src =\":/img/res/tree_r2.png\">"
    "      </figure>"
    "   </li>"
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Если родитель \"y\" есть NULL, сделать \"x\" как
корень дерева."
    "       </font>"
    "       <br>"
    "   </li>"
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Иначе, если \"y\" правый дочерний элемент своего
родителя \"p\", сделать \"x\" как правильный ребенок \"p\""
    "       </font>"
    "       <br>"
    "   </li>"
    "   <li>"
    "       <font style=\"vertical-align: inherit;\">"
    "           Иначе назначить \"x\" как левый ребенок \"p\""
```

```
    "       </font>"
    "       <br>"
    "       <figure>"
    "          <img src =\":/img/res/tree_r3.png\">"
    "       </figure>"
    "   </li>"
    "   <li>"
    "      <font style=\"vertical-align: inherit;\">"
    "          Сделать \"x\" как родитель \"y\""
    "      </font>"
    "       <br>"
    "      <figure>"
    "         <img src =\":/img/res/tree_r4.png\">"
    "      </figure>"
    "   </li>"
    "</ol>");

}

Documentation::~Documentation() {

    delete close;
    delete grid_layout;
    delete info;
}
```

HELPERFUNCTIONS.CPP:

```
#include "helperfunctions.h"
bool isRed(Node *elem)
{
    if (elem != nullptr)
        return elem->getColor() == RED;

    return false;
}

bool isBlack(Node *elem)
{
    if (elem != nullptr){
```

```
        return elem->getColor() == BLACK;
    }

    return true;
}

Node *Parent(Node *elem)
{
    // * Обратите внимание, что для корневого узла родительский
элемент имеет значение null.
    return elem == nullptr ? nullptr : elem->getParent();
}

Node *GrandParent(Node *elem)
{
    // * Обратите внимание, что он вернет nullptr, если это root или
дочерний элемент root
    return Parent(Parent(elem));
}

bool isLeft(Node *elem)
{
    if (Parent(elem))
        return Parent(elem)->getLeft() == elem;

    return false;
}

bool isRight(Node *elem)
{
    if (Parent(elem))
        return Parent(elem)->getRight() == elem;
    return false;
}

Node *Brother(Node *elem){
    // * Отсутствие родителя означает отсутствие брата или сестры.
```

```
    if (Parent(elem))
    {
        if (isLeft(elem))
        {
            return Parent(elem)->getRight();
        }
        else
        {
            return Parent(elem)->getLeft();
        }
    }

    return nullptr;
}

Node *Uncle(Node *elem)
{
    //* Отсутствие родителя означает отсутствие дяди
    return Brother(Parent(elem));
}

Node *Root(Node *elem){

    Node *root = elem;
    if(root){
        while (root->getParent())
        {
            root = root->getParent();
        }
    }
    return root;
}

Node *findElem(Node *root, int key)
{
    Node *tmp = root;
```

```
    while (tmp != nullptr)
    {
        if (tmp->getValue() == key)
        {
            return tmp;
        }
        if (tmp->getValue() > key)
        {
            tmp = tmp->getLeft();
        }
        else
        {
            tmp = tmp->getRight();
        }
    }
    return nullptr;
}

void free(Node *elem){

    if( isLeaf( elem )){

        return;
    }
    if( isLeft ( elem ) ) {

        Parent(elem )->setLeft(nullptr);
    }
    else if( isRight ( elem )  )
    {
        Parent(elem)->setRight(nullptr);
    }

    delete elem;
    elem = nullptr;
}
```

```
int CountElem(Node *elem, int key){

    int l = 0, r = 0;
    if(elem->getLeft())
        l = CountElem(elem->getLeft(), key);

    if(elem->getRight())

        r = CountElem(elem->getRight(), key);

    if( elem->getValue() == key ){
        return 1+l+r;
    }
    return l+r;
}

bool isNotLeaf(Node *elem){
    return elem != nullptr;
}

bool isLeaf(Node *elem){
    return elem == nullptr;
}
```