Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе**

Дисциплина: «Основы теории алгоритмов и структуры данных»

Тема: Бинарные деревья

Выполнил:

Студентка группы РИС-22-1б

Черкасова Алёна Алексеевна

Проверила:

Доцент кафедры ИТАС

Полякова Ольга Андреевна

**Пермь 2023**

Цель работы

Реализовать приложение с пользовательским интерфейсом для работы с бинарным деревом поиска.

Постановка задачи

Требуется реализовать алгоритмы для собственного варианта бинарного дерева поиска, имеющего не менее трёх уровней.

Алгоритмы:

* Необходимо реализовать функции для редактирования дерева:
  + Вставка узла.
  + Удаление узла.
  + Поиск элемента по ключу.
* Реализовать алгоритмы обхода дерева:
  + Прямой
  + Симметричный
  + Обратный
* Реализовать алгоритм балансировки дерева.
* Реализовать вертикальную и горизонтальную печать.
* Визуализация дерева должна быть выполнена с использованием любой доступной графической библиотеки – SFML, SDL, OpenGL и подобных.
* Пользовательский интерфейс по усмотрению разработчика - с условием кроссплатформенности (поощряется использование Qt или иных фреймворков).

Анализ задачи

Разработка проекта реализована на фреймворке Qt.

Проект для возможности взаимодействия бинарного дерева с диалоговыми окнами содержит текстовые файлы added.txt, deleted.txt, finded.txt, size.txt и tree.txt для хранения информации о добавляемом, удаляемом и искомом узлах, размера бинарного дерева и значениях узлов бинарного дерева дереве соответственно.

При создании приложения использовались принципы объектно-ориентированного программирования.

Описание классов

Для реализации приложения созданы следующие классы:

Класс MainWindow описывает главное окно приложения, с помощью которого производятся манипуляции с бинарным деревом.

Главное окно приложения состоит из следующих элементов пользовательского интерфейса:

* Функциональная кнопка “Добавить узел” типа QPushButton обеспечивает выполнение добавления узла из бинарного дерева через соответствующее диалоговое окно.
* Функциональная кнопка “Удалить узел” типа QPushButton обеспечивает выполнение удаления узла из бинарного дерева через соответствующее диалоговое окно.
* Функциональная кнопка “Удалить дерево” типа QPushButton обеспечивает выполнение удаления бинарного дерева.
* Функциональная кнопка “Поиск элемента” типа QPushButton обеспечивает выполнение поиск узла бинарного дерева через соответствующее диалоговое окно.
* Функциональная кнопка “Сбалансировать дерево” типа QPushButton обеспечивает выполнение удаления узла из бинарного дерева.
* Функциональная кнопка “Вертикальная/Горизонтальная печать” типа QPushButton обеспечивает изменение режима визуализации структуры бинарного дерева. В рамка проекта реализованы вертикальная и горизонтальная печать.
* Текстовое поле типа QTextEdit обеспечивает вывод данных, являющихся последовательностью, образуемой прямым обходом бинарного дерева.
* Текстовое поле типа QTextEdit обеспечивает вывод данных, являющихся последовательностью, образуемой симметричным обходом бинарного дерева.
* Текстовое поле типа QTextEdit обеспечивает вывод данных, являющихся последовательностью, образуемой обратным обходом бинарного дерева.
* Канва типа QGraphicView обеспечивает возможность визуализации структуры бинарного дерева

Класс MainWindow содержит следующие методы:

* Метод ButtonAdd реализует открытие диалогового окна для добавления узла при нажатии на кнопку “Добавить узел”.
* Метод ButtonDelete реализует открытие диалогового окна для удаления узла при нажатии на кнопку “Удалить узел”.
* Метод ButtonFind реализует открытие диалогового окна для добавления узла при нажатии на кнопку “Поиск элемента”.
* Метод ButtonClear реализует выполнение удаления бинарного дерева при нажатии на кнопку “Поиск элемента”.
* Метод ButtonBalance реализует выполнение балансировки бинарного дерева при нажатии на кнопку “Поиск элемента”.
* Метод ButtonPrintMode реализует изменение режима визуализации бинарного дерева при нажатии на кнопку “Вертикальная/Горизонтальная печать”.
* Метод EditTree реализует заполнение бинарного дерева значениями из файла хранения
* Метод ClearFiles реализует очищение внешних файлов хранения size.txt и tree.txt.
* Метод ClearText реализует очищение текстовых полей.
* Метод SetText реализует заполнение текстовых полей вариантами вывода обходов бинарного дерева.
* Метод EditText реализует выполнение двух вышеописанных методов ClearText и SetText.
* Метод UpdateScene реализует выполнение визуализации бинарного дерева.

Класс AddWindow описывает диалоговое окно, используемое для добавления узла бинарного дерева.

Диалоговое окно состоит из следующих элементов пользовательского интерфейса:

* Текстовое поле типа QLineEdit обеспечивает пользовательский ввод данных, являющихся значением добавляемого узла бинарного дерева.
* Функциональная кнопка “Добавить узел” типа QPushButton обеспечивает выполнение добавления узла в бинарное дерево.

Класс AddWindow содержит следующие методы:

* Метод AddNode реализует выполнение добавления узла в бинарное дерево при нажатии на кнопку “Добавить узел”.

Класс DeleteWindow описывает диалоговое окно, используемое для удаления узла бинарного дерева по ключу.

Диалоговое окно состоит из следующих элементов пользовательского интерфейса:

* Текстовое поле типа QLineEdit обеспечивает пользовательский ввод данных, являющихся значением удаляемого узла бинарного дерева.
* Функциональная кнопка “Удалить узел” типа QPushButton обеспечивает выполнение удаления узла из бинарного дерева.

Класс DeleteWindow содержит следующие методы:

* Метод DeleteNode реализует выполнение удаления узла из бинарного дерева при нажатии на кнопку “Удалить узел”.

Класс FindWindow описывает диалоговое окно, используемое для поиска узла бинарного дерева по ключу.

Диалоговое окно состоит из следующих элементов пользовательского интерфейса:

* Текстовое поле типа QLineEdit обеспечивает пользовательский ввод данных, являющихся значением искомого узла бинарного дерева.
* Функциональная кнопка “Найти узел” типа QPushButton обеспечивает выполнение поиска узла бинарного дерева по ключу.

Класс AddWindow содержит следующие методы:

* Метод FindNode реализует поиск узла бинарного дерева по ключу при нажатии на кнопку “Найти узел”.

Класс AddErrorDialog описывает вспомогательное диалоговое окно, которое служит для уведомления пользователя об ошибке при добавлении узла.

Диалоговое окно состоит из следующих элементов пользовательского интерфейса:

* Объект типа Label обеспечивает отображение сообщения об ошибке добавления узла в бинарное дерево пользователю.

Класс DeleteErrorDialog описывает вспомогательное диалоговое окно, которое служит для уведомления пользователя об ошибке при удалении узла.

Диалоговое окно состоит из следующих элементов пользовательского интерфейса:

* Объект типа Label обеспечивает отображение сообщения об ошибке удаления узла из бинарного дерева пользователю.

Класс FindNodeDialog описывает диалоговое окно, которое служит для уведомления пользователя о существовании узла с заданным ключом.

Диалоговое окно состоит из следующих элементов пользовательского интерфейса:

* Объект типа Label обеспечивает отображение сообщения о наличии узла в бинарном дереве пользователю.

Класс NodeShape описывает инструментарий для визуализации узла бинарного дерева.

Класс NodeShape содержит следующие атрибуты:

* Публичные атрибуты NodePenColor и NodeBrushColor типа QColor для хранения информации о цветах для визуализации узла.
* Защищённые атрибуты node\_x и node\_y типа double для хранения координат узла.
* Защищённые атрибуты text\_x и text\_y типа double для хранения координат значения узла.
* Защищённый атрибут node\_radius типа double для хранения радиуса узла.

Класс NodeShape содержит следующие методы:

* Публичный метод SetNodeCoords реализует заполнение координат узла.
* Публичный метод SetTextCoords реализует заполнение координат значения узла.
* Публичный метод SetTextCoords реализует заполнение координат значения узла.
* Защищённый виртуальный метод boundingRect реализует выделение области для рисования узла.
* Защищённый метод paint реализует рисование узла.

Класс Node описывает структуру узла бинарного дерева.

Класс Node содержит следующие атрибуты:

* Скрытый атрибут data типа int для хранения значения узла.
* Скрытый указатель на атрибут left типа Node для хранения ссылки на левую ветвь поддерева.
* Скрытый указатель на атрибут right типа Node для хранения ссылки на правую ветвь поддерева.

Класс BinaryTree описывает содержимое и поведение бинарного дерева.

Класс BinaryTree содержит следующие атрибуты:

* Защищённый указатель на атрибут root типа Node для хранения корневого узла бинарного дерева.
* Защищённый указатель на атрибут nodeshape типа NodeShape для хранения визуальной модели узла бинарного дерева.

Класс BinaryTree содержит следующие методы:

* Защищённый метод is\_empty реализует проверку на наличие элементов в бинарном дереве.
* Защищённый вспомогательный метод \_clear реализует очистку бинарного дерева.
* Защищённый вспомогательный метод \_visualise реализует визуализацию бинарного дерева.
* Защищённый вспомогательный метод \_forward реализует создание строки со значениями узлов бинарного дерева, используя прямой обход.
* Защищённый вспомогательный метод \_symmetric реализует создание строки со значениями узлов бинарного дерева, используя симметричный обход.
* Защищённый вспомогательный метод \_reverse реализует создание строки со значениями узлов бинарного дерева, используя обратный обход.
* Защищённый вспомогательный виртуальный метод \_insert реализует алгоритм добавления узла в бинарное дерево.
* Защищённый вспомогательный виртуальный метод \_remove реализует алгоритм удаления узла из бинарного дерева.
* Защищённый вспомогательный виртуальный метод \_contains реализует алгоритм поиска узла бинарного дерева по ключу.
* Защищённый вспомогательный метод \_make\_vector реализует конвертацию бинарного дерева в вектор значений узлов бинарного дерева.
* Защищённый вспомогательный метод \_make\_balanced реализует конвертацию вектора значений узлов бинарного дерева в сбалансированное бинарное дерево.
* Публичный метод clear реализует выполнение вспомогательного метода \_clear.
* Публичный метод print реализует выполнение одного из вспомогательных методов \_forward, \_symmetric или \_reverse.
* Публичный метод visualise реализует выполнение вспомогательного метода \_visualise.
* Публичный виртуальный метод insert реализует выполнение вспомогательного метода \_insert.
* Публичный виртуальный метод remove реализует выполнение вспомогательного метода \_remove.
* Публичный виртуальный метод contains реализует выполнение вспомогательного метода \_contains.
* Публичный метод balance реализует выполнение вспомогательного методов \_make\_vector и \_make\_balanced.

Класс SearchTree является наследуемым от класса BinaryTree и описывает поведение бинарного дерева поиска.

Класс SearchTree содержит следующие атрибуты:

* Скрытый указатель на атрибут minimum типа Node для хранения узла бинарного дерева поиска с минимальным значением.
* Скрытый указатель на атрибут removed типа Node для хранения удаляемого узла бинарного дерева поиска.

Класс SearchTree содержит следующие методы:

* Защищённый вспомогательный переписанный метод \_insert реализует алгоритм добавления узла в бинарное дерево.
* Защищённый вспомогательный переписанный метод \_remove реализует алгоритм удаления узла из бинарного дерева.
* Защищённый вспомогательный метод min реализует алгоритм поиск узла бинарного дерева с минимальным значением.
* Защищённый вспомогательный переписанный метод \_contains реализует алгоритм поиска узла бинарного дерева поиска по ключу.
* Публичный переписанный метод insert реализует выполнение вспомогательного метода \_insert.
* Публичный переписанный метод remove реализует выполнение вспомогательного метода \_remove.
* Публичный переписанный метод contains реализует выполнение вспомогательного метода \_contains.

Реализация вышеописанных классов представлена в приложении (см. Приложение).

UML-диаграмма

Ниже представлена UML-диаграмма проекта (см. Рисунок 1).

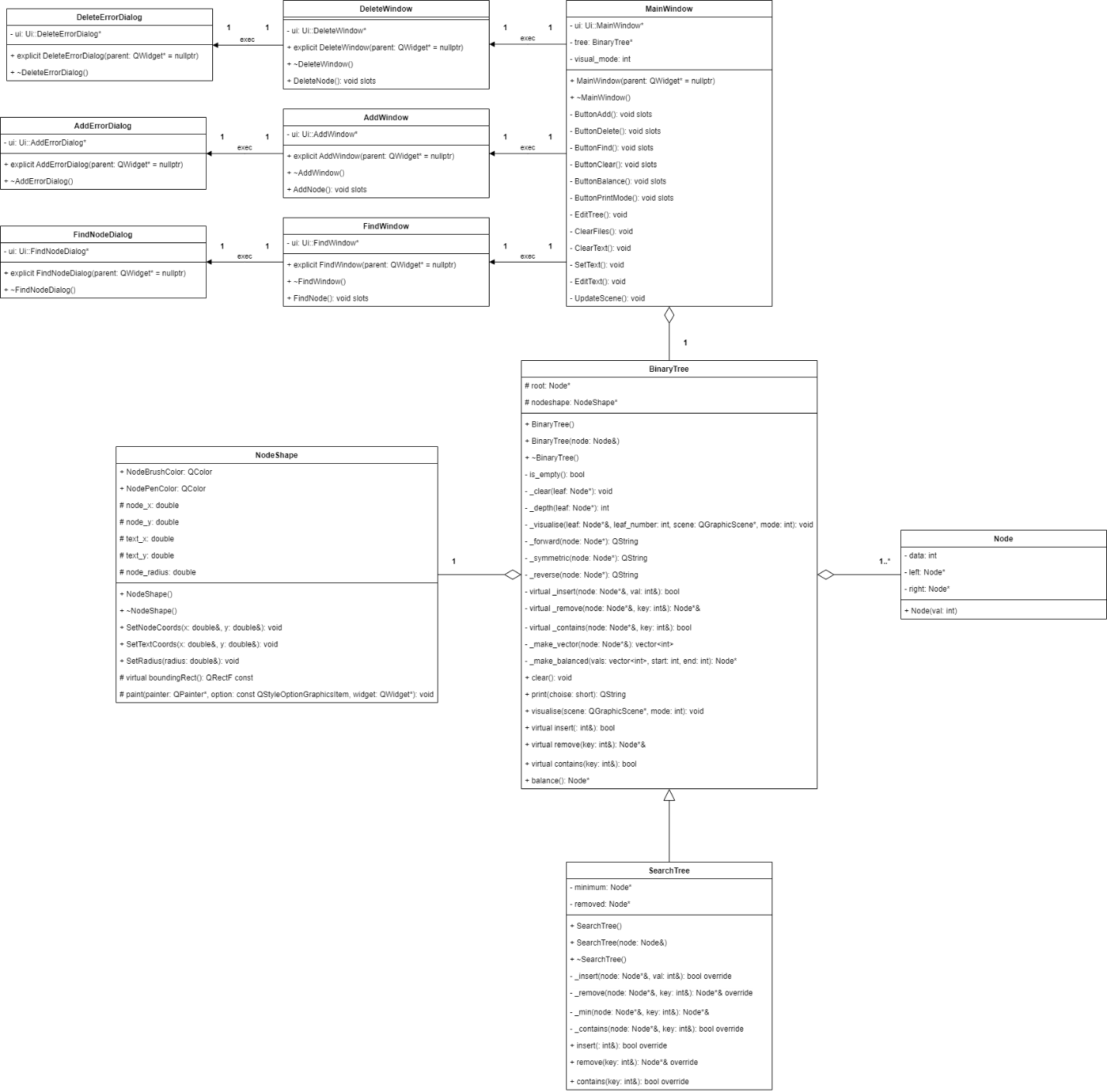


Рисунок 1 – UML-диаграмма проекта GraphicTree

ПРИЛОЖЕНИЕ

Исходный код файла adderrordialog.h

#ifndef ADDERRORDIALOG\_H

#define ADDERRORDIALOG\_H

#include <QDialog>

#include <QFile>

#include <QTextStream>

namespace Ui {

class AddErrorDialog;

}

class AddErrorDialog : public QDialog

{

Q\_OBJECT

public:

explicit AddErrorDialog(QWidget \*parent = nullptr);

~AddErrorDialog();

private:

Ui::AddErrorDialog \*ui;

};

#endif // ADDERRORDIALOG\_H

Исходный код файла adderrordialog.cpp

#include "adderrordialog.h"

#include "ui\_adderrordialog.h"

AddErrorDialog::AddErrorDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::AddErrorDialog)

{

ui->setupUi(this);

QFile added\_error\_file("C:/QtRepos/GraphicTree/added.txt");

QTextStream added\_error\_stream(&added\_error\_file);

added\_error\_file.open(QFile::ReadOnly | QFile::Text);

QString row = "Узел со значением " + added\_error\_stream.readLine() + " уже существует!";

added\_error\_file.flush();

added\_error\_file.close();

ui->label->setText(row);

}

AddErrorDialog::~AddErrorDialog()

{

delete ui;

}

Исходный код файла addwindow.h

#ifndef ADDWINDOW\_H

#define ADDWINDOW\_H

#include <QDialog>

#include <QFile>

#include <QTextStream>

#include "adderrordialog.h"

#include "searchtree.h"

namespace Ui {

class AddWindow;

}

class AddWindow : public QDialog

{

Q\_OBJECT

public:

explicit AddWindow(QWidget \*parent = nullptr);

~AddWindow();

private slots:

// Реализация кнопок

void AddNode();

private:

Ui::AddWindow \*ui;

};

#endif // ADDWINDOW\_H

Исходный код файла addwindow.cpp

#include "addwindow.h"

#include "ui\_addwindow.h"

QFile added\_file("C:/QtRepos/GraphicTree/added.txt"),

tree\_add\_file("C:/QtRepos/GraphicTree/tree.txt"),

size\_add\_file("C:/QtRepos/GraphicTree/size.txt");

QTextStream added\_stream(&added\_file),

tree\_add\_stream(&tree\_add\_file),

size\_add\_stream(&size\_add\_file);

AddWindow::AddWindow(QWidget \*parent) :

QDialog(parent),

ui(new Ui::AddWindow)

{

ui->setupUi(this);

connect(ui->addBtn, &QPushButton::clicked, this, &AddWindow::AddNode);

}

void AddWindow::AddNode()

{

bool success;

int key = ui->addText->text().toInt(&success);

if (ui->addText->text() != "" || success)

{

// Создаём бинарное дерево tree на основе файла tree.txt

size\_add\_file.open(QFile::ReadOnly | QFile::Text);

int size = size\_add\_stream.readLine().toInt();

size\_add\_file.flush();

size\_add\_file.close();

tree\_add\_file.open(QFile::ReadOnly | QFile::Text);

BinaryTree\* temp = new SearchTree();

for (int i = 0; i < size; i++)

{

int val = tree\_add\_stream.readLine().toInt();

temp->insert(val);

}

tree\_add\_file.flush();

tree\_add\_file.close();

// Выполняем нужное нам действие + Обработка исключительной ситуации с помощью диалогового окна

added\_file.open(QFile::WriteOnly | QFile::Text | QFile::Truncate);

added\_stream << QString::number(key) << Qt::endl;

if (!temp->insert(key))

{

added\_file.flush();

added\_file.close();

AddErrorDialog dialog;

dialog.setModal(true);

dialog.exec();

}

else

{

// Перезаполнение файла tree.txt новыми данными дерева (tree->print(1))

size++;

size\_add\_file.open(QFile::WriteOnly | QFile::Text | QFile::Truncate);

size\_add\_stream << QString::number(size);

size\_add\_file.flush();

size\_add\_file.close();

tree\_add\_file.open(QFile::WriteOnly | QFile::Text | QFile::Truncate);

QStringList vals = temp->print(1).split(' '); // Метод возвращает последовательность с доп пробелом в конце

vals.removeLast(); // Поэтому последний элемент удаяется

for (int i = 0; i < vals.size(); i++)

{

tree\_add\_stream << vals[i] << Qt::endl;

}

tree\_add\_file.flush();

tree\_add\_file.close();

}

added\_file.flush();

added\_file.close();

// Удаление бинарного дерева tree

delete temp;

close();

}

ui->addText->clear();

}

AddWindow::~AddWindow()

{

delete ui;

}

Исходный код файла binarytree.h

#ifndef BINARYTREE\_H

#define BINARYTREE\_H

#pragma once

#include <QString>

#include <QGraphicsScene>

#include <algorithm>

#include <iostream>

#include <string>

#include <vector>

#include "nodeshape.h"

using namespace std;

class Node

{

public:

int data;

Node\* left;

Node\* right;

Node(int val);

};

class BinaryTree

{

protected:

Node\* root;

NodeShape\* nodeshape;

bool is\_empty();

void \_clear(Node\* leaf);

int \_depth(Node\* leaf);

void \_visualise(Node\*& leaf, int leaf\_number, QGraphicsScene \*scene, int mode);

QString \_forward(Node\* node);

QString \_symmetric(Node\* node);

QString \_reverse(Node\* node);

virtual bool \_insert(Node\*& node, int& val) = 0;

virtual Node\*& \_remove(Node\*& node, int& key) = 0;

virtual bool \_contains(Node\*& node, int& key) = 0;

vector<int> \_make\_vector(Node\*& node);

Node\* \_make\_balanced(vector<int> vals, int start, int end);

public:

BinaryTree();

BinaryTree(Node& node);

~BinaryTree();

void clear();

QString print(short choice);

void visualise(QGraphicsScene \*scene, int mode);

virtual bool insert(int& val) = 0;

virtual Node\*& remove(int& key) = 0;

virtual bool contains(int& key) = 0;

Node\* balance();

};

#endif // BINARYTREE\_H

Исходный код файла binarytree.cpp

#include "binarytree.h"

#include <QtCore/qmath.h>

//QString line;

/\*==============================\*/

int x, y; // x - номер элемента на уровне, y - номер уровня

int offset = 250; // Отступ от краев сцены

double R = 0; // Рисуемый радиус узла дерева

double node\_x = 0, node\_y = 0; // Координаты узла

double text\_x = 0, text\_y = 0; // Координаты текста

int theight = 0; // Число уровней дерева

double x\_1 = 0, y\_1 = 0, x\_2 = 0, y\_2 = 0; // Начальные и конечные координаты линии

int coefficient; // Коэффициент используется в формуле расчета координат начала линии. Значения равны 1 и -1 в зависимости от того, влево рисуется линия или вправо

double height = 0, width = 0; // Размеры сцены

bool isCurrentNodeLeft; // Флажок нужен для определения значения коэффициента

/\*==============================\*/

Node::Node(int val)

{

{

data = val;

left = nullptr;

right = nullptr;

}

}

bool BinaryTree::is\_empty()

{

return root == nullptr;

}

void BinaryTree::\_clear(Node\* leaf)

{

if (leaf != nullptr)

{

\_clear(leaf->left);

\_clear(leaf->right);

delete leaf;

leaf = nullptr;

}

}

int BinaryTree::\_depth(Node \*leaf)

{

if (leaf != nullptr)

{

return qMax(\_depth(leaf->right), \_depth(leaf->left)) + 1;

}

else

{

return 0;

}

}

void BinaryTree::\_visualise(Node\*& leaf, int leaf\_number, QGraphicsScene\* scene, int mode)

{

if (leaf == nullptr)

{

return;

}

y++;

x = leaf\_number;

R = (width - offset) / qPow(2, theight);

nodeshape = new NodeShape();

node\_x = width \* (2 \* x - 1) / qPow(2, y);

node\_y = height \* (qSqrt(y \* y \* (y \* 1.315))) / qPow(2, theight);

if (y == 1)

{

x\_1 = node\_x;

y\_1 = node\_y;

}

else

{

x\_2 = node\_x;

y\_2 = node\_y;

double D = qSqrt(qPow(x\_2 - x\_1, 2) + qPow(y\_2 - y\_1, 2)); // AC

double d = D - R; // AK

double big\_triangle\_cathetus = qSqrt(qPow(x\_2 - x\_1, 2)); // AB

/\* sinC = AB / AC; sinC = AH / AK \*/

double small\_triangle\_cathetus = big\_triangle\_cathetus \* d / D; // AH

if (isCurrentNodeLeft)

{

coefficient = 1;

}

else

{

coefficient = -1;

}

double new\_x\_1 = x\_2 + coefficient \* small\_triangle\_cathetus; // K(x2 +|AH|; t)

/\* AK = sqrt((x2 + |AH| - x2)^2 + (y2 - t)^2), где t = новые координаты y1 \*/

double new\_y\_1 = y\_2 - qSqrt(d \* d - qPow(small\_triangle\_cathetus, 2)); // t = y2 - sqrt(AK^2-AH^2)

x\_1 = new\_x\_1;

y\_1 = new\_y\_1;

QGraphicsItem\* edge;

if (mode == 0)

{

edge = scene->addLine(x\_1, y\_1, x\_2, y\_2, QPen(Qt::black)); // Ребро

}

else

{

edge = scene->addLine(y\_1, x\_1, y\_2, x\_2, QPen(Qt::black)); // Ребро

}

scene->addItem(edge);

x\_1 = x\_2;

y\_1 = y\_2;

}

if (mode == 0)

{

nodeshape->SetNodeCoords(node\_x, node\_y);

}

else

{

nodeshape->SetNodeCoords(node\_y, node\_x);

}

QString line = QString::number(leaf->data);

text\_x = node\_x - R / 6 \* line.length();

text\_y = node\_y - 3 \* R / 8;

nodeshape->SetNodeRadius(R);

scene->addItem(nodeshape);

nodeshape->setPos(0, 0);

QFont font("Times", R / 3);

QGraphicsTextItem\* text = scene->addText(QString::number(leaf->data), font);

if (mode == 0)

{

text->setPos(text\_x, text\_y);

}

else

{

text->setPos(text\_y, text\_x);

}

isCurrentNodeLeft = true;

\_visualise(leaf->left, 2 \* leaf\_number - 1, scene, mode);

isCurrentNodeLeft = false;

// Находим координаты узла-родителя перед входом в правую ветку

x\_1 = width \* ( 2 \* leaf\_number) / qPow(2, y); // node\_x или leaf\_number

y\_1 = height \* (qSqrt(y \* y \* (y \* 1.315))) / qPow(2, theight); // node\_y

\_visualise(leaf->right, 2 \* leaf\_number, scene, mode);

y--;

if (y == 1)

{

x = 1;

x\_1 = width \* (2 \* x - 1) / qPow(2, y); // node\_x

y\_1 = height \* (qSqrt(y \* y \* (y \* 1.315))) / qPow(2, theight); // node\_y

}

}

QString BinaryTree::\_forward(Node\* node)

{

QString row = "";

if (node != nullptr)

{

row += QString::number(node->data) + " ";

row += \_forward(node->left);

row += \_forward(node->right);

}

return row;

}

QString BinaryTree::\_symmetric(Node\* node)

{

QString row = "";

if (node != nullptr)

{

row += \_symmetric(node->left);

row += QString::number(node->data) + " ";

row += \_symmetric(node->right);

}

return row;

}

QString BinaryTree::\_reverse(Node\* node)

{

QString row = "";

if (node != nullptr)

{

row += \_reverse(node->left);

row += \_reverse(node->right);

row += QString::number(node->data) + " ";

}

return row;

}

vector<int> BinaryTree::\_make\_vector(Node\*& node)

{

vector<int> vals = vector<int>();

if (node != nullptr)

{

vector<int> left = \_make\_vector(node->left);

vector<int> right = \_make\_vector(node->right);

vals.insert(vals.end(), left.begin(), left.end());

vals.push\_back(node->data);

vals.insert(vals.end(), right.begin(), right.end());

}

return vals;

}

Node\* BinaryTree::\_make\_balanced(vector<int> vals, int start, int end)

{

Node\* node = nullptr;

if (start <= end)

{

int mid = (start + end) / 2;

node = new Node(vals[mid]);

node->left = \_make\_balanced(vals, start, mid - 1);

node->right = \_make\_balanced(vals, mid + 1, end);

}

return node;

}

BinaryTree::BinaryTree()

{

root = nullptr;

}

BinaryTree::BinaryTree(Node& node)

{

root = &node;

}

BinaryTree::~BinaryTree()

{

clear();

}

void BinaryTree::clear()

{

\_clear(root);

}

QString BinaryTree::print(short choice)

{

QString row = "";

switch (choice)

{

case 1:

row += \_forward(root);

break;

case 2:

row += \_symmetric(root);

break;

case 3:

row += \_reverse(root);

break;

default:

break;

}

return row;

}

Node\* BinaryTree::balance()

{

vector<int> vals = \_make\_vector(root);

return \_make\_balanced(vals, 0, vals.size() - 1);

}

void BinaryTree::visualise(QGraphicsScene \*scene, int mode)

{

x = 0;

y = 0;

theight = \_depth(root);

width = scene->width();

height = scene->height();

\_visualise(root, 1, scene, mode);

}

Исходный код файла deleteerrordialog.h

#ifndef DELETEERRORDIALOG\_H

#define DELETEERRORDIALOG\_H

#include <QDialog>

#include <QFile>

#include <QTextStream>

namespace Ui {

class DeleteErrorDialog;

}

class DeleteErrorDialog : public QDialog

{

Q\_OBJECT

public:

explicit DeleteErrorDialog(QWidget \*parent = nullptr);

~DeleteErrorDialog();

private:

Ui::DeleteErrorDialog \*ui;

};

#endif // DELETEERRORDIALOG\_H

Исходный код файла deleteerrordialog.cpp

#include "deleteerrordialog.h"

#include "ui\_deleteerrordialog.h"

DeleteErrorDialog::DeleteErrorDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::DeleteErrorDialog)

{

ui->setupUi(this);

QFile deleted\_error\_file("C:/QtRepos/GraphicTree/deleted.txt");

QTextStream deleted\_error\_stream(&deleted\_error\_file);

deleted\_error\_file.open(QFile::ReadOnly | QFile::Text);

QString row = "Узел со значением " + deleted\_error\_stream.readLine() + " не существует!";

deleted\_error\_file.flush();

deleted\_error\_file.close();

ui->label->setText(row);

}

DeleteErrorDialog::~DeleteErrorDialog()

{

delete ui;

}

Исходный код файла deletewinodw.h

#ifndef DELETEWINDOW\_H

#define DELETEWINDOW\_H

#include <QDialog>

#include <QFile>

#include <QTextStream>

#include "deleteerrordialog.h"

#include "searchtree.h"

namespace Ui {

class DeleteWindow;

}

class DeleteWindow : public QDialog

{

Q\_OBJECT

public:

explicit DeleteWindow(QWidget \*parent = nullptr);

~DeleteWindow();

private slots:

// Реализация кнопок

void DeleteNode();

private:

Ui::DeleteWindow \*ui;

};

#endif // DELETEWINDOW\_H

Исходный код файла deletewindow.cpp

#include "deletewindow.h"

#include "ui\_deletewindow.h"

QFile deleted\_file("C:/QtRepos/GraphicTree/deleted.txt"),

tree\_delete\_file("C:/QtRepos/GraphicTree/tree.txt"),

size\_delete\_file("C:/QtRepos/GraphicTree/size.txt");

QTextStream deleted\_stream(&deleted\_file),

tree\_delete\_stream(&tree\_delete\_file),

size\_delete\_stream(&size\_delete\_file);

DeleteWindow::DeleteWindow(QWidget \*parent) :

QDialog(parent),

ui(new Ui::DeleteWindow)

{

ui->setupUi(this);

connect(ui->deleteBtn, &QPushButton::clicked, this, &DeleteWindow::DeleteNode);

}

void DeleteWindow::DeleteNode()

{

bool success;

int key = ui->deleteText->text().toInt(&success);

if (ui->deleteText->text() != "" || success)

{

// Создаём бинарное дерево tree на основе файла tree.txt

size\_delete\_file.open(QFile::ReadOnly | QFile::Text);

int size = size\_delete\_stream.readLine().toInt();

size\_delete\_file.flush();

size\_delete\_file.close();

tree\_delete\_file.open(QFile::ReadOnly | QFile::Text);

BinaryTree\* temp = new SearchTree();

for (int i = 0; i < size; i++)

{

int val = tree\_delete\_stream.readLine().toInt();

temp->insert(val);

}

tree\_delete\_file.flush();

tree\_delete\_file.close();

// Выполняем нужное нам действие + Обработка исключительной ситуации с помощью диалогового окна

deleted\_file.open(QFile::WriteOnly | QFile::Text | QFile::Truncate);

deleted\_stream << QString::number(key);

if (temp->remove(key) == nullptr) //!temp->contains(key)

{

deleted\_file.flush();

deleted\_file.close();

DeleteErrorDialog dialog;

dialog.setModal(true);

dialog.exec();

}

else

{

// Перезаполнение файла tree.txt новыми данными дерева (tree->print(1))

size--;

size\_delete\_file.open(QFile::WriteOnly | QFile::Text | QFile::Truncate);

size\_delete\_stream << QString::number(size);

size\_delete\_file.flush();

size\_delete\_file.close();

tree\_delete\_file.open(QFile::WriteOnly | QFile::Text | QFile::Truncate);

QStringList vals = temp->print(1).split(' '); // Метод возвращает последовательность с доп пробелом в конце

vals.removeLast(); // Поэтому последний элемент удаляется

for (int i = 0; i < vals.size(); i++)

{

tree\_delete\_stream << vals[i] << Qt::endl;

}

tree\_delete\_file.flush();

tree\_delete\_file.close();

}

deleted\_file.flush();

deleted\_file.close();

// Удаление бинарного дерева tree

delete temp;

close();

}

ui->deleteText->clear();

}

DeleteWindow::~DeleteWindow()

{

delete ui;

}

Исходный код файла findnodedialog.h

#ifndef FINDNODEDIALOG\_H

#define FINDNODEDIALOG\_H

#include <QDialog>

#include <QFile>

#include <QTextStream>

namespace Ui {

class FindNodeDialog;

}

class FindNodeDialog : public QDialog

{

Q\_OBJECT

public:

explicit FindNodeDialog(QWidget \*parent = nullptr);

~FindNodeDialog();

private:

Ui::FindNodeDialog \*ui;

};

#endif // FINDNODEDIALOG\_H

Исходный код файла findnodedialog.cpp

#include "findnodedialog.h"

#include "ui\_findnodedialog.h"

FindNodeDialog::FindNodeDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::FindNodeDialog)

{

ui->setupUi(this);

QFile finded\_error\_file("C:/QtRepos/GraphicTree/finded.txt");

QTextStream finded\_error\_stream(&finded\_error\_file);

finded\_error\_file.open(QFile::ReadOnly | QFile::Text);

QString key = finded\_error\_stream.readLine(),

finded = finded\_error\_stream.readLine(),

row;

row += "Узел со значением " + key + " ";

if (finded != "0")

{

row += "найден!";

}

else

{

row += "не найден!";

}

finded\_error\_file.flush();

finded\_error\_file.close();

ui->label->setText(row);

}

FindNodeDialog::~FindNodeDialog()

{

delete ui;

}

Исходный код файла findwindow.h

#ifndef FINDWINDOW\_H

#define FINDWINDOW\_H

#include <QDialog>

#include <QFile>

#include <QTextStream>

#include "findnodedialog.h"

#include "searchtree.h"

namespace Ui {

class FindWindow;

}

class FindWindow : public QDialog

{

Q\_OBJECT

public:

explicit FindWindow(QWidget \*parent = nullptr);

~FindWindow();

private slots:

// Реализация кнопок

void FindNode();

private:

Ui::FindWindow \*ui;

};

#endif // FINDWINDOW\_H

Исходный код файла findwindow.cpp

#include "findwindow.h"

#include "ui\_findwindow.h"

QFile finded\_file("C:/QtRepos/GraphicTree/finded.txt"),

tree\_find\_file("C:/QtRepos/GraphicTree/tree.txt"),

size\_find\_file("C:/QtRepos/GraphicTree/size.txt");

QTextStream finded\_stream(&finded\_file),

tree\_find\_stream(&tree\_find\_file),

size\_find\_stream(&size\_find\_file);

FindWindow::FindWindow(QWidget \*parent) :

QDialog(parent),

ui(new Ui::FindWindow)

{

ui->setupUi(this);

connect(ui->findBtn, &QPushButton::clicked, this, &FindWindow::FindNode);

}

void FindWindow::FindNode()

{

bool success;

int key = ui->findText->text().toInt(&success);

if (ui->findText->text() != "" || success)

{

// Создаём бинарное дерево tree на основе файла tree.txt

tree\_find\_file.open(QFile::ReadOnly | QFile::Text);

BinaryTree\* temp = new SearchTree();

while (!tree\_find\_file.atEnd())

{

int val = tree\_find\_stream.readLine().toInt();

temp->insert(val);

}

tree\_find\_file.flush();

tree\_find\_file.close();

// Выполняем нужное нам действие + Обработка исключительной ситуации с помощью диалогового окна

finded\_file.open(QFile::WriteOnly | QFile::Text | QFile::Truncate);

finded\_stream << QString::number(key) << Qt::endl;

finded\_stream << QString::number(temp->contains(key));

finded\_file.flush();

finded\_file.close();

FindNodeDialog dialog;

dialog.setModal(true);

dialog.exec();

finded\_file.open(QFile::WriteOnly | QFile::Text);

finded\_file.flush();

finded\_file.close();

// Удаление бинарного дерева tree

delete temp;

close();

}

ui->findText->clear();

}

FindWindow::~FindWindow()

{

delete ui;

}

Исходный код файла main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Исходный код файла mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "addwindow.h"

#include "deletewindow.h"

#include "findwindow.h"

#include "searchtree.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void ButtonAdd();

void ButtonDelete();

void ButtonFind();

void ButtonClear();

void ButtonBalance();

void ButtonPrintMode();

private:

void EditTree();

void ClearFiles();

void ClearText();

void SetText();

void EditText();

void UpdateScene();

Ui::MainWindow \*ui;

BinaryTree\* tree;

int visual\_mode;

};

#endif // MAINWINDOW\_H

Исходный код файла mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QPainter>

#include <QGridLayout>

#include <QPushButton>

#include <QProcess>

#include <QFile>

#include <QTextStream>

QFile tree\_global\_file("C:/QtRepos/GraphicTree/tree.txt"),

size\_global\_file("C:/QtRepos/GraphicTree/size.txt");

QTextStream tree\_global\_stream(&tree\_global\_file),

size\_global\_stream(&size\_global\_file);

QGraphicsScene \*scene;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

ClearFiles();

tree = new SearchTree();

visual\_mode = 0;

scene = new QGraphicsScene();

scene->setSceneRect(0, 0, 2000, 2000);

ui->canvas->setScene(scene); // Отображение сцены с элементами

scene->clear();

connect(ui->addNodeBtn, &QPushButton::clicked, this, &MainWindow::ButtonAdd);

connect(ui->deleteNodeBtn, &QPushButton::clicked, this, &MainWindow::ButtonDelete);

connect(ui->findNodeBtn, &QPushButton::clicked, this, &MainWindow::ButtonFind);

connect(ui->deleteTreeBtn, &QPushButton::clicked, this, &MainWindow::ButtonClear);

connect(ui->balanceTreeBtn, &QPushButton::clicked, this, &MainWindow::ButtonBalance);

connect(ui->switchModeBtn, &QPushButton::clicked, this, &MainWindow::ButtonPrintMode);

}

void MainWindow::EditTree()

{

delete tree;

tree = new SearchTree();

size\_global\_file.open(QFile::ReadOnly | QFile::Text);

int size = size\_global\_stream.readLine().toInt();

size\_global\_file.flush();

size\_global\_file.close();

tree\_global\_file.open(QFile::ReadOnly | QFile::Text);

for (int i = 0; i < size; i++)

{

int val = tree\_global\_stream.readLine().toInt();

tree->insert(val);

}

tree\_global\_file.flush();

tree\_global\_file.close();

EditText();

}

void MainWindow::ClearFiles()

{

tree\_global\_file.open(QFile::WriteOnly | QFile::Truncate);

tree\_global\_file.flush();

tree\_global\_file.close();

size\_global\_file.open(QFile::WriteOnly | QFile::Truncate);

size\_global\_stream << QString::number(0);

size\_global\_file.flush();

size\_global\_file.close();

}

void MainWindow::ClearText()

{

ui->forwardText->clear();

ui->symmetricText->clear();

ui->reverseText->clear();

}

void MainWindow::SetText()

{

ui->forwardText->setText(tree->print(1));

ui->symmetricText->setText(tree->print(2));

ui->reverseText->setText(tree->print(3));

}

void MainWindow::EditText()

{

ClearText();

SetText();

}

void MainWindow::UpdateScene()

{

scene->clear();

tree->visualise(scene, visual\_mode);

}

void MainWindow::ButtonAdd()

{

AddWindow window;

window.setModal(true);

window.exec();

EditTree();

UpdateScene();

}

void MainWindow::ButtonDelete()

{

DeleteWindow window;

window.setModal(true);

window.exec();

EditTree();

UpdateScene();

}

void MainWindow::ButtonFind()

{

FindWindow window;

window.setModal(true);

window.exec();

EditText();

}

void MainWindow::ButtonClear()

{

ClearFiles();

delete tree;

tree = new SearchTree();

EditText();

UpdateScene();

}

void MainWindow::ButtonBalance()

{

BinaryTree\* temp = tree;

tree = new SearchTree(\*temp->balance());

delete temp;

tree\_global\_file.open(QFile::WriteOnly | QFile::Text | QFile::Truncate);

QStringList vals = tree->print(1).split(' '); // Метод возвращает последовательность с доп пробелом в конце

vals.removeLast(); // Поэтому последний элемент удаляется

for (int i = 0; i < vals.size(); i++)

{

tree\_global\_stream << vals[i] << Qt::endl;

}

tree\_global\_file.flush();

tree\_global\_file.close();

EditTree();

UpdateScene();

}

void MainWindow::ButtonPrintMode()

{

if (ui->switchModeBtn->text() == "Вертикальная печать")

{

visual\_mode = 1;

ui->switchModeBtn->setText("Горизонтальная печать");

}

else

{

visual\_mode = 0;

ui->switchModeBtn->setText("Вертикальная печать");

}

UpdateScene();

}

MainWindow::~MainWindow()

{

ClearFiles();

delete tree;

delete ui;

}

Исходный код файла nodeshape.h

#ifndef NODESHAPE\_H

#define NODESHAPE\_H

#include <QGraphicsItem>

#include <QPainter>

#include <QColor>

class NodeShape: public QGraphicsItem

{

public:

NodeShape();

~NodeShape();

QColor NodeBrushColor;

QColor NodePenColor;

void SetNodeCoords(double& x, double& y);

void SetTextCoords(double& x, double& y);

void SetNodeRadius(double& radius);

protected:

double node\_x, node\_y;

double text\_x, text\_y;

double node\_radius;

virtual QRectF boundingRect() const; // Выделение области для рисования узла

void paint(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget); // Рисование узла

};

#endif // NODESHAPE\_H

Исходный код файла nodeshape.cpp

#include "nodeshape.h"

NodeShape::NodeShape(): QGraphicsItem()

{

NodeBrushColor = Qt::cyan;

NodePenColor = Qt::black;

}

NodeShape::~NodeShape()

{

}

QRectF NodeShape::boundingRect() const

{

return QRectF(node\_x - node\_radius, node\_y - node\_radius, node\_radius \* 2, node\_radius \* 2);

}

void NodeShape::SetNodeCoords(double& x, double& y)

{

node\_x = x;

node\_y = y;

}

void NodeShape::SetTextCoords(double& x, double& y)

{

text\_x = x;

text\_y = y;

}

void NodeShape::SetNodeRadius(double& radius)

{

node\_radius = radius;

}

void NodeShape::paint(QPainter \*painter, const QStyleOptionGraphicsItem\* option, QWidget\* widget)

{

QPointF pointF(node\_x, node\_y);

painter->setBrush(NodeBrushColor);

painter->setPen(NodePenColor);

painter->drawEllipse(pointF, node\_radius, node\_radius);

Q\_UNUSED(option);

Q\_UNUSED(widget);

}

Исходный код файла searchtree.h

#ifndef SEARCHTREE\_H

#define SEARCHTREE\_H

#pragma once

#include "binarytree.h"

class SearchTree : public BinaryTree

{

private:

Node\* minimum;

Node\* removed;

bool \_insert(Node\*& node, int& val) override;

Node\*& \_remove(Node\*& node, int& key) override;

Node\*& \_min(Node\*& node);

bool \_contains(Node\*& node, int& key) override;

public:

SearchTree();

SearchTree(Node& node);

~SearchTree();

bool insert(int& val) override;

Node\*& remove(int& key) override;

bool contains(int& key) override;

};

#endif // SEARCHTREE\_H

Исходный код файла searchtree.cpp

#include "searchtree.h"

bool SearchTree::\_insert(Node\*& node, int& val)

{

bool added = false;

if (node == nullptr)

{

node = new Node(val);

added = true;

}

else

{

if (val < node->data)

{

added = \_insert(node->left, val);

}

else if (val > node->data)

{

added = \_insert(node->right, val);

}

else

{

added = false;

}

}

return added;

}

Node\*& SearchTree::\_remove(Node\*& node, int& key)

{

removed = nullptr;

if (node != nullptr)

{

if (node->data == key)

{

if (node->left == nullptr and node->right == nullptr)

{

removed = node;

node = nullptr;

}

else if (node->left != nullptr and node->right == nullptr)

{

removed = node;

node = node->left;

}

else if (node->left == nullptr and node->right != nullptr)

{

removed = node;

node = node->right;

}

else if (node->left != nullptr and node->right != nullptr)

{

Node\*& minimum = \_min(node->right);

node->data = minimum->data;

minimum->data = key;

removed = \_remove(minimum, minimum->data);

}

}

else

{

if (key < node->data)

{

removed = \_remove(node->left, key);

}

else

{

removed = \_remove(node->right, key);

}

}

}

return removed;

}

Node\*& SearchTree::\_min(Node\*& node)

{

minimum = nullptr;

if (node != nullptr)

{

if (node->left == nullptr)

{

minimum = node;

}

else

{

minimum = \_min(node->left);

}

}

return minimum;

}

bool SearchTree::\_contains(Node\*& node, int& key)

{

bool exists = false;

if (node != nullptr)

{

if (key < node->data)

{

exists = \_contains(node->left, key);

}

else if (key > node->data)

{

exists = \_contains(node->right, key);

}

else

{

exists = true;

}

}

return exists;

}

SearchTree::SearchTree() : BinaryTree() {}

SearchTree::SearchTree(Node& node) : BinaryTree(node) {}

SearchTree::~SearchTree() {}

bool SearchTree::insert(int& val)

{

return \_insert(root, val);

}

Node\*& SearchTree::remove(int& key)

{

return \_remove(root, key);

}

bool SearchTree::contains(int& key)

{

return \_contains(root, key);

}