**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **Рандомизированные пирамиды поиска**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Сергеев А.Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

## Цель работы.

Ознакомиться с основными характеристиками и особенностями рандомизированных пирамид поиска, изучить особенности их реализации на языке программирования C++. Разработать программу, демонстрирующую принцип работы пирамид.

## Задание.

- По заданному файлу F (типа file of Elem), все элементы которого различны, построить структуру данных определённого типа – БДП.

- Реализовать сцепление двух пирамид.

## Основные теоретические положения.

Рандомизированная пирамида поиска — это бинарное дерево, в узлах которого хранятся пары (x,y), где x — это ключ, а y — это приоритет. Также оно является двоичным деревом поиска по x и пирамидой по y. Предполагая, что все x и все y являются различными, получаем, что если некоторый элемент дерева содержит (x0,y0), то у всех элементов в левом поддереве x<x0, у всех элементов в правом поддереве x>x0, а также и в левом, и в правом поддереве имеем: y<y0.

При каждой вставке в пирамиду очередного элемента ключ выбирается случайно из чисел, принадлежащих определённому отрезку.

## Выполнение работы.

Написание работы производилось на базе операционной системы *Ubuntu*, в среде *CLion*, а также с использованием библиотек *qt* и среды *QTCreator*.

Для выполнения поставленной задачи был создан класс *rand\_tree*, содержащий в себе методы работы с бинарным деревом поиска, также был создан класс *tree\_node*, в котором была реализована вставка узлов в пирамиду при помощи двух методов: *split* и *merge*.

*Split* позволяет разрезать дерево по ключу *k*, так, что в одной половине все ключи будут меньше *k*, а в другой — больше. С помощью метода merge можно слить два дерева в одно (но только при условии, что все ключи первого поддерева будут меньше ключей правого).

Класс *lab5* содержит в себе все методы для выполнения задания: построение пирамиды (в быстром и пошаговом режиме, с данными, введёнными в строку и в файл), слияние двух пирамид (в быстром и пошаговом режиме). Реализован пошаговый режим через простейший стэк (класс *stack*) на базе списка.

Для изображения пирамиды на экране используется класс *myglwidget*, наследующий от класса *qopenglwidget*, и содержащий два режима отрисовки дерева — с узлами и без. Режим отрисовки выбирается в зависимости от глубины дерева. Для наглядности узел, в котором было сделано последнее изменение подсвечивается красным (или, когда работа с пирамидой закончена, красным подсвечивается корень).

Дополнительно реализован ещё один, тестовый режим работы с пирамидой, проводящий построение 10000 раз и выводящий на экран среднюю и идеальную высоту.

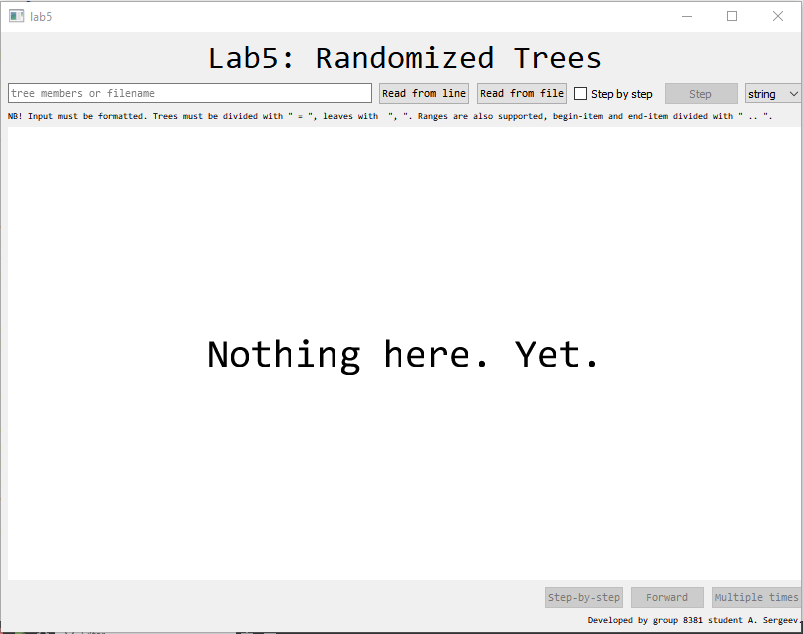
Исходный код программы см. в Приложении А.

## Оценка эффективности алгоритма.

Все методы работы с пирамидой реализованы через вставку узлов в пирамиду. Вставка узлов состоит из двух методов merge и одного split. Как split, так и merge теоретически требуют O(n) операций, где n – высота дерева, так как в них рекурсивно вызывается O(1) операций для каждого дерева меньшей высоты. Операция слияния двух пирамид (ключи в которых не сортированы) требует рекурсивного обхода второй пирамиды, что, теоретически требует O(k) операций, где k – количество элементов во второй пирамиде.

**Тестирование программы.**

Ниже представлены снимки экрана работающей в режиме *gui* программы, а также результаты трёх различных тестов.



## 

## 

## 

## 

## Входные данные: «1 .. 3 = 4 .. 7», тип: *int*.

## Выводы.

В ходе выполнения лабораторной работы была изучена такая структура данных как пирамида, а также методы её обработки. Была реализована программа на С++, демонстрирующая принцип работы рандомизированных пирамид поиска.

# Приложение А Исходный код программы

**Файл main.cpp:**

#include "mainwindow.h"

#include <QApplication>

#include <QtQuick>

#include <sstream>

#include <iostream>

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

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**Файл lab5.h:**

#ifndef LAB5\_H

#define LAB5\_H

#include "rand\_tree.h"

#include "myglwidget.h"

#include <iostream>

#include <fstream>

#include <sstream>

class lab5 {

public:

static int launch(MyGLWidget\* mglw, std::string\* input, bool ff, int tp, bool step);

template<typename T>

static int launch\_steps(MyGLWidget\* mglw) {

if (!tree1\_stck<T>->isEmpty()) {

auto val = tree1\_stck<T>->pop();

tree1<T>->add(val);

mglw->set\_tree(tree1<T>, val);

mglw->show(&output\_file);

return 1;

} else if (!tree2\_stck<T>->isEmpty()) {

auto val = tree2\_stck<T>->pop();

tree2<T>->add(val);

mglw->set\_tree(tree2<T>, val);

mglw->show(&output\_file);

return 1;

} else {

free(tree1\_stck<T>);

tree1\_stck<T> = nullptr;

free(tree2\_stck<T>);

tree2\_stck<T> = nullptr;

mglw->set\_tree(tree2<T>, tree2<T>->get\_root()->get\_state());

mglw->show(&output\_file);

return 0;

}

}

static int type;

template<typename T>

static rand\_tree<T>\* tree1;

template<typename T>

static rand\_tree<T>\* tree2;

template<typename T>

static int step(MyGLWidget\* mglw) {

if (stck<T> == nullptr) {

stck<T> = new stack<tree\_node<T>\*>();

stck<T>->push(tree2<T>->get\_root());

mglw->set\_tree(tree1<T>, tree1<T>->get\_root()->get\_state());

mglw->show(&output\_file);

return 1;

} else if (!stck<T>->isEmpty()) {

auto node = stck<T>->pop();

tree1<T>->join(node, stck<T>);

tree1<T>->get\_root()->reset\_weight();

mglw->set\_tree(tree1<T>, node->get\_state());

mglw->show(&output\_file);

free(node);

return 1;

} else {

free(stck<T>);

stck<T> = nullptr;

mglw->set\_tree(tree1<T>, tree1<T>->get\_root()->get\_state());

mglw->show(&output\_file);

return 0;

}

}

template<typename T>

static int rush(MyGLWidget\* mglw) {

tree1<T>->join(tree2<T>->get\_root(), nullptr);

mglw->set\_tree(tree1<T>, tree1<T>->get\_root()->get\_state());

mglw->show(&output\_file);

return 0;

}

template<typename T>

static int mult(MyGLWidget\* mglw) {

rand\_tree<T>\* new\_tree = nullptr;

long long int mid\_size = 0;

long long int theo\_size = 0;

for (int i = 0; i < 10000; i++) {

auto tree\_string = new std::string(input\_str);

new\_tree = new rand\_tree<T>(tree\_string, nullptr);

if (theo\_size == 0) theo\_size = new\_tree->theory\_depth();

mid\_size += new\_tree->max\_depth();

free(new\_tree);

free(tree\_string);

new\_tree = nullptr;

}

std::stringstream ss;

ss << "Average depth of 10000 trees was " << mid\_size / 10000 << "\nIdeal size is " << theo\_size;

auto msg = new std::string(ss.str());

mglw->declare(msg);

return 0;

}

private:

static const std::string input\_file;

static const std::string output\_file;

static std::string input\_str;

template<typename T>

static stack<tree\_node<T>\*>\* stck;

template<typename T>

static stack<T>\* tree1\_stck;

template<typename T>

static stack<T>\* tree2\_stck;

};

template<typename T>

stack<tree\_node<T>\*>\* lab5::stck = nullptr;

template<typename T>

stack<T>\* lab5::tree1\_stck = nullptr;

template<typename T>

stack<T>\* lab5::tree2\_stck = nullptr;

template<typename T>

rand\_tree<T>\* lab5::tree1 = nullptr;

template<typename T>

rand\_tree<T>\* lab5::tree2 = nullptr;

#endif // LAB5\_H

**Файл lab5.cpp:**

#include "lab5.h"

const std::string lab5::input\_file = "C:/Users/miles/Documents/lab5/in.txt";

const std::string lab5::output\_file = "C:/Users/miles/Documents/lab5/out.png";

const std::string tree\_divider = " = ";

int lab5::type = 0;

std::string lab5::input\_str = "";

int lab5::launch(MyGLWidget\* mglw, std::string\* input, bool ff, int tp, bool step) {

mglw->prepare\_drawing();

std::string tree\_string;

if (ff) {

std::ifstream is;

if (input->empty()) {

is = std::ifstream(input\_file);

} else {

is = std::ifstream(\*input);

}

if (is && !is.fail()) {

getline(is, tree\_string);

} else {

auto msg = new std::string("File can not be opened :(");

mglw->declare(msg);

return 1;

}

} else {

if (input->empty()) {

auto msg = new std::string("The input was empty.");

mglw->declare(msg);

return 1;

} else {

tree\_string = \*input;

}

}

unsigned long long div = tree\_string.find(tree\_divider);

if (div == std::string::npos) {

auto msg = new std::string("String for only one tree was provided.");

mglw->declare(msg);

return 1;

}

std::string first = tree\_string.substr(0, div);

std::string second = tree\_string.substr(div + tree\_divider.length(), tree\_string.length() - 1);

if (first.empty() || second.empty()) {

auto msg = new std::string("The input for one of the trees was empty.");

mglw->declare(msg);

return 1;

}

input\_str = tree\_string.replace(div, tree\_divider.length(), ", ");

type = tp;

try {

switch (type) {

case 0: {

if (step) {

tree1\_stck<std::string> = new stack<std::string>();

tree2\_stck<std::string> = new stack<std::string>();

}

tree1<std::string> = new rand\_tree<std::string>(&first, tree1\_stck<std::string>);

tree2<std::string> = new rand\_tree<std::string>(&second, tree2\_stck<std::string>);

break;

}

case 1: {

if (step) {

tree1\_stck<char> = new stack<char>();

tree2\_stck<char> = new stack<char>();

}

tree1<char> = new rand\_tree<char>(&first, tree1\_stck<char>);

tree2<char> = new rand\_tree<char>(&second, tree2\_stck<char>);

break;

}

case 2: {

if (step) {

tree1\_stck<int> = new stack<int>();

tree2\_stck<int> = new stack<int>();

}

tree1<int> = new rand\_tree<int>(&first, tree1\_stck<int>);

tree2<int> = new rand\_tree<int>(&second, tree2\_stck<int>);

break;

}

case 3: {

if (step) {

tree1\_stck<double> = new stack<double>();

tree2\_stck<double> = new stack<double>();

}

tree1<double> = new rand\_tree<double>(&first, tree1\_stck<double>);

tree2<double> = new rand\_tree<double>(&second, tree2\_stck<double>);

break;

}

}

} catch (std::runtime\_error re) {

auto msg = new std::string(re.what());

mglw->declare(msg);

return 1;

}

std::string\* msg;

if (step) {

msg = new std::string("Both strings successfully loaded. We are ready 2 build.");

} else {

msg = new std::string("Both trees successfully loaded. We are ready 2 go.");

}

mglw->declare(msg);

return ((step) ? (2) : (0));

}

**Файл mainwindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private:

Ui::MainWindow \*ui;

void enable\_input(bool val);

void step\_input(bool val);

private slots:

void read\_from\_file();

void read\_from\_line();

void step();

void step\_forward();

void rush\_forward();

void mult\_forward();

};

#endif // MAINWINDOW\_H

**Файл mainwindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "lab5.h"

MainWindow::MainWindow(QWidget \*parent) : QMainWindow(parent), ui(new Ui::MainWindow) {

ui->setupUi(this);

QStringList available\_types = {"string", "char", "int", "double"};

ui->type\_selector->addItems(available\_types);

connect(ui->line\_button, SIGNAL(clicked()), this, SLOT(read\_from\_line()));

connect(ui->file\_button, SIGNAL(clicked()), this, SLOT(read\_from\_file()));

connect(ui->step\_button, SIGNAL(clicked()), this, SLOT(step()));

connect(ui->straightforward\_button, SIGNAL(clicked()), this, SLOT(rush\_forward()));

connect(ui->proceed\_button, SIGNAL(clicked()), this, SLOT(step\_forward()));

connect(ui->multiple\_button, SIGNAL(clicked()), this, SLOT(mult\_forward()));

}

MainWindow::~MainWindow() {

delete ui;

}

void MainWindow::enable\_input(bool val) {

ui->straightforward\_button->setEnabled(!val);

ui->proceed\_button->setEnabled(!val);

ui->multiple\_button->setEnabled(!val);

ui->line\_button->setEnabled(val);

ui->file\_button->setEnabled(val);

ui->type\_selector->setEnabled(val);

ui->line\_input->setEnabled(val);

ui->step\_box->setEnabled(val);

}

void MainWindow::step\_input(bool val) {

ui->straightforward\_button->setEnabled(!val);

ui->proceed\_button->setEnabled(!val);

ui->multiple\_button->setEnabled(!val);

ui->step\_button->setEnabled(val);

}

void MainWindow::read\_from\_file() {

std::string input = ui->line\_input->text().toStdString();

int result = lab5::launch(ui->canvas, &input, true, ui->type\_selector->currentIndex(), ui->step\_box->isChecked());

if (result == 0) {

enable\_input(false);

ui->answer->setText("Both trees successfully loaded.");

} else if (result == 2) {

enable\_input(false);

step\_input(true);

ui->answer->setText("Continue building...");

} else {

ui->answer->setText("Error occures, input again.");

}

}

void MainWindow::read\_from\_line() {

std::string input = ui->line\_input->text().toStdString();

int result = lab5::launch(ui->canvas, &input, false, ui->type\_selector->currentIndex(), ui->step\_box->isChecked());

if (result == 0) {

enable\_input(false);

ui->answer->setText("Both trees successfully loaded.");

} else if (result == 2) {

enable\_input(false);

step\_input(true);

ui->answer->setText("Continue building...");

} else {

ui->answer->setText("Error occures, input again.");

}

}

void MainWindow::step() {

int result = 1;

switch (lab5::type) {

case 0:

result = lab5::launch\_steps<std::string>(ui->canvas);

break;

case 1:

result = lab5::launch\_steps<char>(ui->canvas);

break;

case 2:

result = lab5::launch\_steps<int>(ui->canvas);

break;

case 3:

result = lab5::launch\_steps<double>(ui->canvas);

break;

}

if (result == 0) {

step\_input(false);

ui->answer->setText("Both trees successfully loaded.");

} else {

ui->answer->setText("Continue building...");

}

}

void MainWindow::step\_forward() {

ui->straightforward\_button->setEnabled(false);

ui->multiple\_button->setEnabled(false);

int result = 1;

switch (lab5::type) {

case 0:

result = lab5::step<std::string>(ui->canvas);

break;

case 1:

result = lab5::step<char>(ui->canvas);

break;

case 2:

result = lab5::step<int>(ui->canvas);

break;

case 3:

result = lab5::step<double>(ui->canvas);

break;

}

if (result == 0) {

enable\_input(true);

ui->answer->setText("Merge successful.");

} else {

ui->answer->setText("Continue merging...");

}

}

void MainWindow::rush\_forward() {

int result = 1;

switch (lab5::type) {

case 0:

result = lab5::rush<std::string>(ui->canvas);

break;

case 1:

result = lab5::rush<char>(ui->canvas);

break;

case 2:

result = lab5::rush<int>(ui->canvas);

break;

case 3:

result = lab5::rush<double>(ui->canvas);

break;

}

if (result == 0) {

enable\_input(true);

ui->answer->setText("Merge successful.");

} else {

ui->answer->setText("Error occures, merge again.");

}

}

void MainWindow::mult\_forward() {

int result = 1;

switch (lab5::type) {

case 0:

result = lab5::mult<std::string>(ui->canvas);

break;

case 1:

result = lab5::mult<char>(ui->canvas);

break;

case 2:

result = lab5::mult<int>(ui->canvas);

break;

case 3:

result = lab5::mult<double>(ui->canvas);

break;

}

if (result == 0) {

enable\_input(true);

ui->answer->setText("Research successful.");

} else {

ui->answer->setText("Error occures, research again.");

}

}

**Файл mainwindow.ui:**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>600</height>

</rect>

</property>

<property name="toolTipDuration">

<number>5</number>

</property>

<widget class="QWidget" name="central\_widget">

<property name="sizePolicy">

<sizepolicy hsizetype="Expanding" vsizetype="Expanding">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<widget class="QWidget" name="verticalLayoutWidget">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>801</width>

<height>601</height>

</rect>

</property>

<property name="sizePolicy">

<sizepolicy hsizetype="Expanding" vsizetype="Expanding">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<layout class="QVBoxLayout" name="container">

<property name="leftMargin">

<number>7</number>

</property>

<property name="topMargin">

<number>7</number>

</property>

<property name="rightMargin">

<number>7</number>

</property>

<property name="bottomMargin">

<number>7</number>

</property>

<item>

<widget class="QLabel" name="name">

<property name="sizePolicy">

<sizepolicy hsizetype="Preferred" vsizetype="Minimum">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<property name="font">

<font>

<family>Consolas</family>

<pointsize>24</pointsize>

</font>

</property>

<property name="text">

<string>Lab5: Randomized Trees</string>

</property>

<property name="alignment">

<set>Qt::AlignCenter</set>

</property>

</widget>

</item>

<item>

<layout class="QHBoxLayout" name="input\_container">

<item>

<widget class="QLineEdit" name="line\_input">

<property name="font">

<font>

<family>Consolas</family>

</font>

</property>

<property name="placeholderText">

<string>tree members or filename</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="line\_button">

<property name="font">

<font>

<family>Consolas</family>

</font>

</property>

<property name="text">

<string>Read from line</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="file\_button">

<property name="font">

<font>

<family>Consolas</family>

</font>

</property>

<property name="text">

<string>Read from file</string>

</property>

</widget>

</item>

<item>

<widget class="QCheckBox" name="step\_box">

<property name="text">

<string>Step by step</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="step\_button">

<property name="enabled">

<bool>false</bool>

</property>

<property name="text">

<string>Step</string>

</property>

</widget>

</item>

<item>

<widget class="QComboBox" name="type\_selector"/>

</item>

</layout>

</item>

<item>

<widget class="QLabel" name="label">

<property name="font">

<font>

<family>Consolas</family>

<pointsize>7</pointsize>

</font>

</property>

<property name="text">

<string>NB! Input must be formatted. Trees must be divided with &quot; = &quot;, leaves with &quot;, &quot;. Ranges are also supported, begin-item and end-item divided with &quot; .. &quot;.</string>

</property>

</widget>

</item>

<item>

<widget class="MyGLWidget" name="canvas">

<property name="sizePolicy">

<sizepolicy hsizetype="Preferred" vsizetype="Expanding">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

</widget>

</item>

<item>

<layout class="QHBoxLayout" name="output\_container">

<item>

<widget class="QLabel" name="answer">

<property name="sizePolicy">

<sizepolicy hsizetype="Expanding" vsizetype="Preferred">

<horstretch>0</horstretch>

<verstretch>0</verstretch>

</sizepolicy>

</property>

<property name="font">

<font>

<family>Consolas</family>

<pointsize>10</pointsize>

</font>

</property>

<property name="alignment">

<set>Qt::AlignCenter</set>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="proceed\_button">

<property name="enabled">

<bool>false</bool>

</property>

<property name="font">

<font>

<family>Consolas</family>

</font>

</property>

<property name="text">

<string>Step-by-step</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="straightforward\_button">

<property name="enabled">

<bool>false</bool>

</property>

<property name="font">

<font>

<family>Consolas</family>

</font>

</property>

<property name="text">

<string>Forward</string>

</property>

</widget>

</item>

<item>

<widget class="QPushButton" name="multiple\_button">

<property name="enabled">

<bool>false</bool>

</property>

<property name="font">

<font>

<family>Consolas</family>

</font>

</property>

<property name="text">

<string>Multiple times</string>

</property>

</widget>

</item>

</layout>

</item>

<item>

<widget class="QLabel" name="subscription">

<property name="font">

<font>

<family>Consolas</family>

<pointsize>7</pointsize>

</font>

</property>

<property name="text">

<string>Developed by group 8381 student A. Sergeev.</string>

</property>

<property name="alignment">

<set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set>

</property>

</widget>

</item>

</layout>

</widget>

</widget>

</widget>

<customwidgets>

<customwidget>

<class>MyGLWidget</class>

<extends>QOpenGLWidget</extends>

<header location="global">myglwidget.h</header>

</customwidget>

</customwidgets>

<resources/>

<connections/>

</ui>

**Файл myglwidget.cpp:**

#include "myglwidget.h"

MyGLWidget::MyGLWidget(QWidget \*parent) : QOpenGLWidget(parent) {}

MyGLWidget::~MyGLWidget() {}

void MyGLWidget::paintGL() {

if (map != nullptr) {

QPainter painter(this);

painter.drawPixmap(this->rect(), \*map, map->rect());

} else {

QPainter painter(this);

painter.fillRect(this->rect(), QBrush(Qt::white));

painter.setPen(Qt::black);

painter.setFont(QFont("Consolas", 30));

painter.drawText(rect(), Qt::AlignCenter, "Nothing here. Yet.");

}

}

void MyGLWidget::show(const std::string\* outfile) {

this->update();

QFile file(QString::fromStdString(\*outfile));

file.open(QIODevice::WriteOnly);

map->save(&file, "PNG");

}

**Файл myglwidget.h:**

#ifndef MYGLWIDGET\_H

#define MYGLWIDGET\_H

#include <QOpenGLWidget>

#include <QPainter>

#include <QFile>

#include <iostream>

#include <sstream>

#include "rand\_tree.h"

const int BONES\_LIMIT = 6;

class MyGLWidget : public QOpenGLWidget {

public:

QPixmap\* map = nullptr;

MyGLWidget(QWidget \*parent);

~MyGLWidget() override;

template<typename T>

void set\_tree(rand\_tree<T>\* source, T outline) {

pntr = new QPainter(map);

pntr->fillRect(map->rect(), QBrush(Qt::white));

pntr->setBrush(Qt::white);

pntr->setFont(QFont("Consolas", 8));

boolean drawing\_bones = source->max\_depth() >= BONES\_LIMIT;

if (!drawing\_bones) pntr->setPen(QPen(Qt::black, 3));

unsigned long depth = source->max\_depth();

stepY = static\_cast<unsigned int>(map->height()) / static\_cast<unsigned int>(depth);

paint\_node(source->get\_root()->get\_left(), false, map->width() / 2, stepY / 2, map->width() / 4, outline);

paint\_node(source->get\_root()->get\_right(), true, map->width() / 2, stepY / 2, map->width() / 4, outline);

if (!drawing\_bones) {

std::stringstream knot;

knot << source->get\_root()->get\_state() << "\n" << source->get\_root()->get\_index();

if (source->get\_root()->get\_state() == outline) {

pntr->setBrush(Qt::red);

pntr->drawEllipse(QPointF(map->width() / 2, stepY / 2), 48, 48);

pntr->setBrush(Qt::white);

} else {

pntr->drawEllipse(QPointF(map->width() / 2, stepY / 2), 48, 48);

}

pntr->drawText(QRect(map->width() / 2 - 24, stepY / 2 - 24, 48, 48), Qt::AlignCenter, QString::fromStdString(knot.str()));

populate\_node(source->get\_root()->get\_left(), false, map->width() / 2, stepY / 2, map->width() / 4, outline);

populate\_node(source->get\_root()->get\_right(), true, map->width() / 2, stepY / 2, map->width() / 4, outline);

}

stepY = 0;

pntr->end();

free(pntr);

}

void declare(std::string\* msg) {

pntr = new QPainter(map);

pntr->fillRect(map->rect(), QBrush(Qt::white));

pntr->setPen(Qt::black);

pntr->setFont(QFont("Consolas", 12));

pntr->drawText(rect(), Qt::AlignCenter, QString::fromStdString(\*msg));

pntr->end();

free(pntr);

this->update();

}

void prepare\_drawing() {

free(this->map);

this->map = new QPixmap(this->width(), this->height());

}

void show(const std::string\* outfile);

protected:

void paintGL() override;

private:

unsigned int stepY = 0;

QPainter\* pntr;

template<typename T>

void paint\_node(tree\_node<T>\* node, boolean is\_right, int parent\_x, int parent\_y, int half\_stepX, T outline) {

if (node == nullptr) return;

int child\_x = parent\_x + ((is\_right) ? (half\_stepX) : (-half\_stepX));

int child\_y = parent\_y + stepY;

if (node->get\_state() == outline) {

pntr->setPen(QPen(Qt::red, pntr->pen().width()));

pntr->drawLine(parent\_x, parent\_y, child\_x, child\_y);

pntr->setPen(QPen(Qt::black, pntr->pen().width()));

} else {

pntr->drawLine(parent\_x, parent\_y, child\_x, child\_y);

}

paint\_node(node->get\_left(), false, child\_x, child\_y, half\_stepX / 2, outline);

paint\_node(node->get\_right(), true, child\_x, child\_y, half\_stepX / 2, outline);

}

template<typename T>

void populate\_node(tree\_node<T>\* node, boolean is\_right, int parent\_x, int parent\_y, int half\_stepX, T outline) {

if (node == nullptr) return;

int child\_x = parent\_x + ((is\_right) ? (half\_stepX) : (-half\_stepX));

int child\_y = parent\_y + stepY;

std::stringstream knot;

knot << node->get\_state() << "\n" << node->get\_index();

if (node->get\_state() == outline) {

pntr->setBrush(Qt::red);

pntr->drawEllipse(QPointF(child\_x, child\_y), 48, 48);

pntr->setBrush(Qt::white);

} else {

pntr->drawEllipse(QPointF(child\_x, child\_y), 48, 48);

}

pntr->drawText(QRect(child\_x - 24, child\_y - 24, 48, 48), Qt::AlignCenter, QString::fromStdString(knot.str()));

if (node->get\_left() != nullptr) {

populate\_node(node->get\_left(), false, child\_x, child\_y, half\_stepX / 2, outline);

}

if (node->get\_right() != nullptr) {

populate\_node(node->get\_right(), true, child\_x, child\_y, half\_stepX / 2, outline);

}

}

};

#endif // MYGLWIDGET\_H

**Файл rand\_tree.h:**

#ifndef CURR\_RAND\_TREE\_H

#define CURR\_RAND\_TREE\_H

#include <cmath>

#include <experimental/type\_traits>

#include <stdexcept>

#include <sstream>

#include "tree\_node.h"

template<typename C>

class rand\_tree {

private:

tree\_node<C>\* root = nullptr;

public:

unsigned long max\_depth();

unsigned long theory\_depth();

unsigned long get\_weight();

tree\_node<C>\* get\_root();

tree\_node<C>\*\* get\_bi\_root();

void join(tree\_node<C>\* node, stack<tree\_node<C>\*>\* stck);

void add(C value);

explicit rand\_tree();

explicit rand\_tree(std::string\* tree\_string, stack<C>\* stck);

};

template<typename C>

using less\_than\_t = decltype(std::declval<C>() < std::declval<C>());

template<typename C>

using more\_than\_t = decltype(std::declval<C>() > std::declval<C>());

template<typename C>

constexpr bool comparable = std::experimental::is\_detected<less\_than\_t, C>::value && std::experimental::is\_detected<more\_than\_t, C>::value;

template<typename C>

unsigned long rand\_tree<C>::max\_depth() {

return root->depth(root);

}

template<typename C>

unsigned long rand\_tree<C>::theory\_depth() {

unsigned long weight = get\_weight();

unsigned long grade = 0;

unsigned int power = 0;

while (grade < weight) {

power++;

grade = static\_cast<unsigned long>(pow(2, power) - 1);

}

return power;

}

template<typename C>

unsigned long rand\_tree<C>::get\_weight() {

return root->get\_weight();

}

template<typename C>

tree\_node<C>\* rand\_tree<C>::get\_root() {

return root;

}

template<typename C>

tree\_node<C>\*\* rand\_tree<C>::get\_bi\_root() {

return &root;

}

template<typename C>

void rand\_tree<C>::join(tree\_node<C>\* node, stack<tree\_node<C>\*>\* stck) {

if (root == nullptr) {

root = node;

} else {

root = tree\_node<C>::insert(root, node->trim());

if (stck == nullptr) {

if (node->get\_left() != nullptr) join(node->get\_left(), stck);

if (node->get\_right() != nullptr) join(node->get\_right(), stck);

} else {

if (node->get\_right() != nullptr) stck->push(node->get\_right());

if (node->get\_left() != nullptr) stck->push(node->get\_left());

}

}

}

template<typename C>

void rand\_tree<C>::add(C value) {

auto node = new tree\_node<C>(value);

if (root == nullptr) {

root = node;

} else {

if (!tree\_node<C>::is(root, value)) {

root = tree\_node<C>::insert(root, node);

root->reset\_weight();

}

}

}

template<typename C>

rand\_tree<C>::rand\_tree() {

if constexpr(!comparable<C>){

throw std::runtime\_error("Tree nodes can not be compared by the key!");

}

}

template<typename C>

rand\_tree<C>::rand\_tree(std::string\* tree\_string, stack<C>\* stck) {

if constexpr(!comparable<C>){

throw std::runtime\_error("Tree nodes can not be compared by the key!");

}

try {

std:: string delimiter = ", ";

std::string range = " .. ";

size\_t pos = 0;

size\_t erase\_pos = 0;

std::string token;

do {

pos = tree\_string->find(delimiter);

erase\_pos = ((pos == std::string::npos) ? tree\_string->length() : pos);

token = tree\_string->substr(0, pos);

unsigned long tos = token.find(range);

unsigned long npos = std::string::npos;

if (tos != npos) {

if (std::is\_same<C, std::string>::value) throw std::runtime\_error("Ranges are not available for strings yet.");

C first, second;

std::stringstream first\_part(token.substr(0, tos));

std::stringstream second\_part(token.substr(tos + range.length(), token.length() - 1));

first\_part >> first;

second\_part >> second;

for (C i = first; i <= second; i += 1) {

if (stck == nullptr) {

this->add(i);

} else {

stck->push(i);

}

}

} else {

C number;

std::stringstream num(token);

std::string str = num.str();

num >> number;

if (stck == nullptr) {

this->add(number);

} else {

stck->push(number);

}

}

tree\_string->erase(0, erase\_pos + delimiter.length());

} while (tree\_string->length() != 0);

} catch (std::runtime\_error re) {

throw std::runtime\_error(re.what());

} catch (...) {

throw std::runtime\_error("Tree string contains error. Tree can not be built.");

}

}

#endif //CURR\_RAND\_TREE\_H

**Файл stack.h:**

#ifndef STACK\_H

#define STACK\_H

#include <cstdlib>

#include <stdexcept>

template <typename T>

class stack {

private:

class stack\_element {

private:

T value;

stack\_element\* next;

stack\_element\* previous;

public:

stack\_element(T& value, stack\_element \*previous);

virtual ~stack\_element();

T& getValue();

void setNext(stack\_element \*next);

stack\_element \*getNext();

stack\_element \*getPrevious();

};

int size;

stack\_element\* first;

stack\_element\* last;

public:

stack();

~stack();

void push(T element);

T pop();

bool isEmpty();

const int\* getStackSize() const;

};

template<typename T>

stack<T>::stack\_element::stack\_element(T& value, stack::stack\_element\* previous) {

this->value = value;

this->previous = previous;

this->next = nullptr;

}

template<typename T>

stack<T>::stack\_element::~stack\_element() {

free(this->next);

free(this->previous);

}

template<typename T>

T& stack<T>::stack\_element::getValue() {

return value;

}

template<typename T>

void stack<T>::stack\_element::setNext(stack::stack\_element \*next) {

stack\_element::next = next;

}

template <typename T>

typename stack<T>::stack\_element \*stack<T>::stack\_element::getNext() {

return this->next;

}

template <typename T>

typename stack<T>::stack\_element \*stack<T>::stack\_element::getPrevious() {

return this->previous;

}

template<typename T>

const int\* stack<T>::getStackSize() const {

return &size;

}

template<typename T>

stack<T>::stack() {

this->size = 0;

this->first = nullptr;

this->last = nullptr;

}

template<typename T>

stack<T>::~stack() {

free(this->first);

free(this->last);

}

template<typename T>

void stack<T>::push(T element) {

stack\_element\* SE = new stack\_element(element, last);

if (size == 0) {

first = SE;

} else {

last->setNext(SE);

}

size++;

last = SE;

}

template<typename T>

T stack<T>::pop() {

if (size == 0)

throw std::runtime\_error("Stack is empty!");

stack\_element \*decapitation = last;

T value = decapitation->getValue();

last = decapitation->getPrevious();

if (size > 1) {

last->setNext(nullptr);

} else {

first = nullptr;

}

free(decapitation);

size--;

return value;

}

template<typename T>

bool stack<T>::isEmpty() {

return size == 0;

}

#endif // STACK\_H

**Файл tree\_node.h:**

#ifndef CURR\_TREE\_NODE\_H

#define CURR\_TREE\_NODE\_H

#include "stack.h"

#include <cstdlib>

#include <random>

template<typename C>

class tree\_node {

private:

C comparable;

double index;

unsigned long weight = 1;

tree\_node\* left = nullptr;

tree\_node\* right = nullptr;

public:

unsigned long get\_weight();

void reset\_weight();

unsigned long depth(tree\_node<C>\* node);

tree\_node\* get\_right() {

return right;

}

tree\_node\* get\_left() {

return left;

}

C get\_state() {

return comparable;

}

double get\_index() {

return index;

}

tree\_node<C>\* trim();

static bool is(tree\_node<C>\* node, C elem);

static void split(tree\_node<C>\* root, C comp, tree\_node<C>\*\* left, tree\_node<C>\*\* right);

static tree\_node<C>\* merge( tree\_node<C>\* first, tree\_node<C>\* second);

static tree\_node<C>\* insert(tree\_node<C>\* first, tree\_node<C>\* second);

explicit tree\_node(C base);

tree\_node();

};

template<typename C>

unsigned long tree\_node<C>::get\_weight() {

return weight;

}

template<typename C>

void tree\_node<C>::reset\_weight() {

unsigned long right\_weight = 0;

unsigned long left\_weight = 0;

if (right != nullptr) {

right->reset\_weight();

right\_weight = right->weight;

}

if (left != nullptr) {

left->reset\_weight();

left\_weight = left->weight;

}

weight = right\_weight + left\_weight + 1;

}

template<typename C>

unsigned long tree\_node<C>::depth(tree\_node<C> \*node) {

if (node == nullptr) {

return 0;

} else {

unsigned long left\_depth = depth(node->left);

unsigned long right\_depth = depth(node->right);

if (left\_depth > right\_depth) {

return (left\_depth + 1);

} else {

return (right\_depth + 1);

}

}

}

template<typename C>

tree\_node<C>\* tree\_node<C>::trim() {

auto node = new tree\_node<C>();

\*node = \*this;

node->left = node->right = nullptr;

return node;

}

template<typename C>

bool tree\_node<C>::is(tree\_node<C>\* node, C elem) {

if (!node) return false;

if (elem == node->get\_state())

return true;

if (elem < node->get\_state())

return is(node->get\_left(), elem);

else

return is(node->get\_right(), elem);

}

template<typename C>

void tree\_node<C>::split(tree\_node<C> \*root, C comp, tree\_node<C> \*\*left, tree\_node<C> \*\*right) {

if (root == nullptr) {

\*left = nullptr;

\*right = nullptr;

return;

} else if (comp > root->get\_state()) {

split(root->right, comp, left, right);

root->right = \*left;

\*left = root;

return;

} else {

split(root->left, comp, left, right);

root->left = \*right;

\*right = root;

return;

}

}

template<typename C>

tree\_node<C> \*tree\_node<C>::merge(tree\_node<C> \*first, tree\_node<C> \*second) {

if (second == nullptr) return first;

if (first == nullptr) return second;

if (first->index > second->index) {

first->right = merge(first->right, second);

return first;

} else {

second->left = merge(first, second->left);

return second;

}

}

template<typename C>

tree\_node<C> \*tree\_node<C>::insert(tree\_node<C>\* first, tree\_node<C>\* second) {

if (first == nullptr) {

return second;

}

if (second == nullptr) {

return first;

}

auto left\_tree = new tree\_node();

auto right\_tree = new tree\_node();

split(first, second->comparable, &left\_tree, &right\_tree);

left\_tree = merge(left\_tree, second);

return merge(left\_tree, right\_tree);

}

template<typename C>

tree\_node<C>::tree\_node(C base) {

this->comparable = base;

this->left = this->right = nullptr;

this->weight = 1;

this->index = ((double) rand() / (RAND\_MAX)) + 1;

}

template<typename C>

tree\_node<C>::tree\_node() {

this->left = this->right = nullptr;

this->weight = 1;

}

#endif //CURR\_TREE\_NODE\_H

**Файл lab5.pro:**

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets quick

CONFIG += c++17

# The following define makes your compiler emit warnings if you use

# any Qt feature that has been marked deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

main.cpp \

mainwindow.cpp \

myglwidget.cpp \

lab5.cpp

HEADERS += \

lab5.h \

mainwindow.h \

myglwidget.h \

stack.h \

tree\_node.h \

rand\_tree.h

FORMS += \

mainwindow.ui

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target