**«Пермский национальный исследовательский  
политехнический университет»**

**О Т Ч Ё Т**

**по лабораторной работе №20**  
Тема: Бинарные деревья

Вариант 5

Выполнил работу

студент группы ИВТ-20-2б

Нефедов Л.В.

Проверила

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**

1) Сформировать идеально сбалансированное дерево с типом информационного поля char

2) Распечатать полученное дерево

3) Найти количество вхождений введённого пользователем элемента

4) Преобразовать идеально сбалансированное дерево в дерево поиска

5) Распечатать полученное дерево

**Анализ задачи**

Для реализации задачи необходимо:

1. Запросить у пользователя ввод количества узлов в дереве.
2. Реализовать ввод значений элементов дерева в массив.
3. Построить идеально сбалансированное дерево.
4. Перестроить идеально сбалансированное дерево в идеально сбалансированное дерево поиска.
5. Найти количество вхождений элемента, введённого пользователем.

**Консольная реализация**

1. Ввод значений элементов дерева в массив:

void add\_tree(int size) {

double number;

cout << "1) ";

cin >> number;

hey = new Mytree;

hey->data = number;

array\_numbers[0] = number;

if ((size - 1) > 0) {

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, left\_subtree\_size, right\_subtree\_size);

add\_tree(hey->left, left\_subtree\_size);

add\_tree(hey->right, right\_subtree\_size);

}

void add\_tree(Mytree\*& leaf, int size) {

if (size == 0) {

leaf = NULL;

return;

}

cout << index + 1 << ") ";

double number;

cin >> number;

array\_numbers[index] = number;

index++;

leaf = new Mytree;

leaf->data = number;

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, left\_subtree\_size, right\_subtree\_size);

add\_tree(leaf->left, left\_subtree\_size);

add\_tree(leaf->right, right\_subtree\_size);

}

1. Алгоритм построения идеально сбалансированного дерева:

while (current!=NULL){

if (current->left!=NULL&&current->left->kras==0){

current=current->left;

glubina++;

}

else if (current->right!=NULL&&current->right->kras==0){

current=current->right;

glubina++;

}

else{

current->kras=1;

pos.X=current->x;

pos.Y=current->y;

SetConsoleCursorPosition(hCon,pos);

cout << s[Stepen(glubina)+number[glubina]-1];

number[glubina]++;

glubina--;

current=current->top;

}

}

1. Алгоритм вывода идеально сбалансированного дерева поиска:

1.int depth\_tree(Mytree\* leaf) {

if (leaf == NULL)

return 0;

return max(depth\_tree(leaf->right), depth\_tree(leaf->left)) + 1;

}

2.void print\_tree(Mytree\*& leaf) {

if (leaf == NULL)

return;

Mytree\* branch\_left = NULL;

Mytree\* branch\_right = NULL;

branch\_left = leaf->left;

branch\_right = leaf->right;

position.X = start\_x\_position;

position.Y = 3;

SetConsoleCursorPosition(hConsole, position);

cout << leaf->data;

int depthOfTheTree = depth\_tree(leaf);

int offset\_coefficient;

if (isBalancedSortPrinting)

offset\_coefficient = 6;

else

offset\_coefficient = 4;

print\_branch(branch\_left, start\_x\_position - offset\_coefficient \* depthOfTheTree,

print\_branch(branch\_right, start\_x\_position + offset\_coefficient \* depthOfTheTree, 5);

}

Mytree\* insert\_in\_search\_tree(Mytree\*& leaf, const double element) {

if (isElementInTree(leaf, element)) //если элемент дублируется, то в дерево поиска мы его не вставляем

return leaf;

if (leaf == NULL) {

Mytree\* new\_leaf = new Mytree;

new\_leaf->data = element;

new\_leaf->left = NULL;

new\_leaf->right = NULL;

return new\_leaf;

}

if (element < leaf->data)

leaf->left = insert\_in\_search\_tree(leaf->left, element);

if (element > leaf->data)

leaf->right = insert\_in\_search\_tree(leaf->right, element);

return leaf;

}

1. Алгоритм нахождения количества лепестков:

void preorder\_metod(Mytree\* leaf) {

if (leaf == NULL)

return;

if (leaf->left == NULL && leaf->right == NULL) i++;

preorder\_metod(leaf->left);

preorder\_metod(leaf->right);

}

**Полный код программы**

Описание класса

#include <iostream>

#include <stdio.h>

#include <Windows.h>

using namespace std;

COORD position;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

int i = 0;

double\* array\_numbers;

int array\_numbers\_size;

int index = 1;

bool isBalancedSortPrinting = true;

const int start\_x\_position = 100;

struct Mytree {

double data;

Mytree\* left, \* right;

};

class BinaryTree

{

private:

Mytree\* hey;

void del\_tree(Mytree\* leaf) {

if (leaf != NULL) {

del\_tree(leaf->left);

del\_tree(leaf->right);

delete leaf;

}

}

bool isElementInTree(Mytree\* leaf, const double& el) {

if (leaf == NULL)

return false;

if (leaf->data == el)

return true;

if (el < leaf->data)

isElementInTree(leaf->left, el);

return isElementInTree(leaf->right, el);

}

void get\_subtrees\_sizes(const int size, int& left\_size, int& right\_size) {

if ((size - 1) % 2 == 0) {

left\_size = (size - 1) / 2;

right\_size = left\_size;

}

else {

left\_size = size / 2;

right\_size = (size - 1) / 2;

}

}

void add\_tree(Mytree\*& leaf, int size) {

if (size == 0) {

leaf = NULL;

return;

}

cout << index + 1 << ") ";

double number;

cin >> number;

array\_numbers[index] = number;

index++;

leaf = new Mytree;

leaf->data = number;

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, left\_subtree\_size, right\_subtree\_size);

add\_tree(leaf->left, left\_subtree\_size);

add\_tree(leaf->right, right\_subtree\_size);

}

void preorder\_metod(Mytree\* leaf) {

if (leaf == NULL)

return;

if (leaf->left == NULL && leaf->right == NULL) i++;

preorder\_metod(leaf->left);

preorder\_metod(leaf->right);

}

int depth\_tree(Mytree\* leaf) {

if (leaf == NULL)

return 0;

return max(depth\_tree(leaf->right), depth\_tree(leaf->left)) + 1;

}

void print\_branch(Mytree\* leaf, int x, int y) {

if (leaf == NULL)

return;

position.X = x;

position.Y = y;

SetConsoleCursorPosition(hConsole, position);

cout << leaf->data;

int formula\_new\_x\_left\_leaf, formula\_new\_x\_right\_leaf;

if (isBalancedSortPrinting) {

formula\_new\_x\_left\_leaf = x - 2 \* (y + 2);

formula\_new\_x\_right\_leaf = x + 2 \* (y + 2);

}

else {

formula\_new\_x\_left\_leaf = x - 2 \* (y + 1);

formula\_new\_x\_right\_leaf = x + 2 \* y;

}

print\_branch(leaf->left, formula\_new\_x\_left\_leaf, y + 2);

print\_branch(leaf->right, formula\_new\_x\_right\_leaf, y + 2);

}

void print\_tree(Mytree\*& leaf) {

if (leaf == NULL)

return;

Mytree\* branch\_left = NULL;

Mytree\* branch\_right = NULL;

branch\_left = leaf->left;

branch\_right = leaf->right;

position.X = start\_x\_position;

position.Y = 3;

SetConsoleCursorPosition(hConsole, position);

cout << leaf->data;

int depthOfTheTree = depth\_tree(leaf);

int offset\_coefficient;

if (isBalancedSortPrinting)

offset\_coefficient = 6;

else

offset\_coefficient = 4;

print\_branch(branch\_left, start\_x\_position - offset\_coefficient \* depthOfTheTree,

print\_branch(branch\_right, start\_x\_position + offset\_coefficient \* depthOfTheTree, 5);

}

Mytree\* insert\_in\_search\_tree(Mytree\*& leaf, const double element) {

if (isElementInTree(leaf, element)) //если элемент дублируется, то в дерево поиска мы его не вставляем

return leaf;

if (leaf == NULL) {

Mytree\* new\_leaf = new Mytree;

new\_leaf->data = element;

new\_leaf->left = NULL;

new\_leaf->right = NULL;

return new\_leaf;

}

if (element < leaf->data)

leaf->left = insert\_in\_search\_tree(leaf->left, element);

if (element > leaf->data)

leaf->right = insert\_in\_search\_tree(leaf->right, element);

return leaf;

}

public:

BinaryTree() {

hey = NULL;

}

~BinaryTree() {

del\_tree(hey);

}

void add\_tree(int size) {

double number;

cout << "1) ";

cin >> number;

hey = new Mytree;

hey->data = number;

array\_numbers[0] = number;

if ((size - 1) > 0) {

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, left\_subtree\_size, right\_subtree\_size);

add\_tree(hey->left, left\_subtree\_size);

add\_tree(hey->right, right\_subtree\_size);

}

}

void preorder\_metod() { //прямой обход

preorder\_metod(hey);

}

int depth\_tree() {

return depth\_tree(hey);

}

void print\_tree() {

print\_tree(hey);

}

void insert\_in\_search\_tree(const double element) {

hey = insert\_in\_search\_tree(hey, element);

}

};

Описание main

#include <iostream>

#include "BinaryTree.h"

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Введите количество элементов в бинарном дереве:" << endl;

do {

cout << ">";

cin >> array\_numbers\_size;

if (array\_numbers\_size < 1)

cout << "Введите положительное число" << endl;

} while (array\_numbers\_size < 1);

array\_numbers = new double[array\_numbers\_size];

BinaryTree tree;

cout << "Введите вещественные числа:\n";

tree.add\_tree(array\_numbers\_size);

system("cls");

cout << "\t\t\t\t\t\t\t\t\t\t\tВЫВОД БИНАРНОГО ДЕРЕВА";

cout << "\n\t\t\t\t\t\t\t\t\t\t\tГлубина дерева: " << tree.depth\_tree();

tree.print\_tree();

cout << "\nКоличество листьев в дереве:" << endl;

tree.preorder\_metod();

cout << i;

cout << endl;

system("pause");

return 0;

}

**Результат работы программы**



**Реализация на базе Qt**

1. Ввод значений и создание сбалансированного дерева:

void BinaryTree::**create\_balanced\_binary\_tree**(int size){

inputFile.*open*(QFile::ReadOnly | QFile::Text);

line = stream.readLine();

root = new Node;

root->data = line.toDouble();

root->left = nullptr;

root->right = nullptr;

line = stream.readLine();

if ((size - 1) > 0) {

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, *left\_subtree\_size*, *right\_subtree\_size*);

create\_balanced\_binary\_tree(*root->left*, left\_subtree\_size);

create\_balanced\_binary\_tree(*root->right*, right\_subtree\_size);

}

inputFile.flush();

inputFile.*close*();

}

1. Функция, идеально балансирующая узлы дерева:

void BinaryTree::**create\_balanced\_binary\_tree**(Node \*&leaf, int size){

if (size == 0) {

leaf = nullptr;

return;

}

if (line == "")

return;

leaf = new Node;

leaf->data = line.toDouble();

leaf->left = nullptr;

leaf->right = nullptr;

line = stream.readLine();

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, *left\_subtree\_size*, *right\_subtree\_size*);

create\_balanced\_binary\_tree(*leaf->left*, left\_subtree\_size);

create\_balanced\_binary\_tree(*leaf->right*, right\_subtree\_size);

}

void BinaryTree::**get\_subtrees\_sizes**(const int size, int &left\_size, int &right\_size){

if ((size - 1) % 2 == 0) {

left\_size = (size - 1) / 2;

right\_size = left\_size;

}

else {

left\_size = size / 2;

right\_size = (size - 1) / 2;

}

}

1. Функция, рисующая дерево в окне программы :

void BinaryTree::**print\_tree**(Node \*&leaf, QGraphicsScene \*scene){

if (leaf == nullptr) {

array\_for\_counting\_indexes[y] = array\_for\_counting\_indexes[y] + 1;

return;

}

y++;

array\_for\_counting\_indexes[y-1] = array\_for\_counting\_indexes[y-1] + 1;

x = array\_for\_counting\_indexes[y - 1];

R = (window\_width - shift)/(qPow(2, tree\_height));

nodeshape = new NodeShape;

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

node\_y = window\_height\*(qSqrt(y\*y\*(y\*1.315)))/qPow(2, tree\_height);

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), where t = new coord of 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 = scene->addLine(x\_1, y\_1, x\_2, y\_2, QPen(Qt::black)); //ребро

scene->addItem(*edge*);

x\_1 = x\_2;

y\_1 = y\_2;

}

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

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);

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

isCurrentNodeLeft = true;

print\_tree(*leaf->left*, *scene*);

isCurrentNodeLeft = false;

//нахожу координаты узла-родителя перед входом в правую веточку

x\_1 = window\_width\*(2\*array\_for\_counting\_indexes[y - 1]-1)/qPow(2, y);//node\_x ||array\_for\_counting\_indexes[y-1]

y\_1 = window\_height\*(qSqrt(y\*y\*(y\*1.315)))/qPow(2, tree\_height);//node\_y

print\_tree(*leaf->right*, *scene*);

y--;

if (y == 1) {

x = 1;

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

y\_1 = window\_height\*(qSqrt(y\*y\*(y\*1.315)))/qPow(2, tree\_height);//node\_y

}

}

**Решение**

1. Заголовочные файлы:

#ifndef BINARYTREE\_H

#define BINARYTREE\_H

#pragma once

#include <QFile>

#include <QGraphicsScene>

#include "nodeshape.h"

struct **Node** {

double data;

Node\* left, \* right;

};

class **BinaryTree**

{

private:

Node\* root;

void **delete\_tree**(Node\* leaf);

void **get\_subtrees\_sizes**(const int size, int& left\_size, int& right\_size);

void **create\_balanced\_binary\_tree**(Node\*& leaf, int size);

int **depth\_tree**(Node\* leaf);

void **print\_tree**(Node\*& leaf, QGraphicsScene \*scene);

public:

NodeShape \*nodeshape;

**BinaryTree**();

~**BinaryTree**();

void **create\_balanced\_binary\_tree**(int size);

int **depth\_tree**();

void **print\_tree**(QGraphicsScene \*scene);

};

#endif // BINARYTREE\_H

#ifndef NODESHAPE\_H

#define NODESHAPE\_H

#include <QGraphicsItem>

#include <QPainter>

#include <QColor>

class **NodeShape**: public QGraphicsItem

{

public:

**NodeShape**();

~***NodeShape***();

QColor NodeBrushColor = Qt::cyan;

QColor NodePenColor = Qt::black;

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;

//выделение области для рисования узла

virtual QRectF ***boundingRect***() const;

//рисование узла

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

//обработчики события наведения мышки:

//вхождение курсора в область прорисовки узла

void ***hoverEnterEvent***(QGraphicsSceneHoverEvent \*event);

//выход курсора из области прорисовки узла

void ***hoverLeaveEvent***(QGraphicsSceneHoverEvent \*event);

};

#endif // NODESHAPE\_H

#ifndef WIDGET\_H

#define WIDGET\_H

#pragma once

#include <QWidget>

#include <QGraphicsScene>

#include <QGraphicsItem>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **Widget**; }

QT\_END\_NAMESPACE

class **Widget** : public QWidget

{

Q\_OBJECT

public:

**Widget**(QWidget \*parent = nullptr);

~***Widget***();

private slots:

void **on\_inputDataBtn\_clicked**();

void **on\_createBalancedTreeBtn\_clicked**();

void **on\_clearSceneBtn\_clicked**();

public:

Ui::Widget \*ui;

};

1. Файл-источники:

#include "binarytree.h"

#include "widget.h"

#include <QFile>

#include <QTextStream>

#include <QtCore/qmath.h>

#include <QFont>

QFile inputFile("C:/BinaryTree/elements.txt");

QTextStream stream(*&inputFile*);

QString line;

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

int shift = 250; //отступ от краев сцены

double R = 0;

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

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

int tree\_height = 0;

int\* array\_for\_counting\_indexes; //хранение количества элементов на уровне. При обращении к массиву достается номер текущего элемента

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 window\_height = 0, window\_width = 0; //размеры сцены

bool isCurrentNodeLeft; //флажок нужен для определения значения коэффициента

NodeShape \*nodeshape;

BinaryTree::**BinaryTree**()

{

root = nullptr;

}

BinaryTree::~**BinaryTree**(){

delete\_tree(*root*);

}

void BinaryTree::**delete\_tree**(Node \*leaf){

if (leaf != nullptr) {

delete\_tree(*leaf->left*);

delete\_tree(*leaf->right*);

delete leaf;

}

}

void BinaryTree::**get\_subtrees\_sizes**(const int size, int &left\_size, int &right\_size){

if ((size - 1) % 2 == 0) {

left\_size = (size - 1) / 2;

right\_size = left\_size;

}

else {

left\_size = size / 2;

right\_size = (size - 1) / 2;

}

}

void BinaryTree::**create\_balanced\_binary\_tree**(Node \*&leaf, int size){

if (size == 0) {

leaf = nullptr;

return;

}

if (line == "")

return;

leaf = new Node;

leaf->data = line.toDouble();

leaf->left = nullptr;

leaf->right = nullptr;

line = stream.readLine();

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, *left\_subtree\_size*, *right\_subtree\_size*);

create\_balanced\_binary\_tree(*leaf->left*, left\_subtree\_size);

create\_balanced\_binary\_tree(*leaf->right*, right\_subtree\_size);

}

int BinaryTree::**depth\_tree**(Node \*leaf){

if (leaf == nullptr)

return 0;

return qMax(depth\_tree(*leaf->right*), depth\_tree(*leaf->left*)) + 1;

}

void BinaryTree::**print\_tree**(Node \*&leaf, QGraphicsScene \*scene){

if (leaf == nullptr) {

array\_for\_counting\_indexes[y] = array\_for\_counting\_indexes[y] + 1;

return;

}

y++;

array\_for\_counting\_indexes[y-1] = array\_for\_counting\_indexes[y-1] + 1;

x = array\_for\_counting\_indexes[y - 1];

R = (window\_width - shift)/(qPow(2, tree\_height));

nodeshape = new NodeShape;

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

node\_y = window\_height\*(qSqrt(y\*y\*(y\*1.315)))/qPow(2, tree\_height);

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), where t = new coord of y1\*/

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

x\_1 = new\_x\_1;

y\_1 = new\_y\_1;

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

scene->addItem(*edge*);

x\_1 = x\_2;

y\_1 = y\_2;

}

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

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);

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

isCurrentNodeLeft = true;

print\_tree(*leaf->left*, *scene*);

isCurrentNodeLeft = false;

//нахожу координаты узла-родителя перед входом в правую веточку

x\_1 = window\_width\*(2\*array\_for\_counting\_indexes[y - 1]-1)/qPow(2, y);//node\_x ||array\_for\_counting\_indexes[y-1]

y\_1 = window\_height\*(qSqrt(y\*y\*(y\*1.315)))/qPow(2, tree\_height);//node\_y

print\_tree(*leaf->right*, *scene*);

y--;

if (y == 1) {

x = 1;

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

y\_1 = window\_height\*(qSqrt(y\*y\*(y\*1.315)))/qPow(2, tree\_height);//node\_y

}

}

void BinaryTree::**create\_balanced\_binary\_tree**(int size){

inputFile.*open*(QFile::ReadOnly | QFile::Text);

line = stream.readLine();

root = new Node;

root->data = line.toDouble();

root->left = nullptr;

root->right = nullptr;

line = stream.readLine();

if ((size - 1) > 0) {

int left\_subtree\_size, right\_subtree\_size;

get\_subtrees\_sizes(size, *left\_subtree\_size*, *right\_subtree\_size*);

create\_balanced\_binary\_tree(*root->left*, left\_subtree\_size);

create\_balanced\_binary\_tree(*root->right*, right\_subtree\_size);

}

inputFile.flush();

inputFile.*close*();

}

int BinaryTree::**depth\_tree**(){

return depth\_tree(*root*);

}

void BinaryTree::**print\_tree**(QGraphicsScene \*scene){

x = 0;

y = 0;

tree\_height = depth\_tree(*root*);

window\_width = scene->width();

window\_height = scene->height();

array\_for\_counting\_indexes = new int[tree\_height+1];

for (int i = 0; i < tree\_height; i++)

array\_for\_counting\_indexes[i] = 0;

print\_tree(*root*, *scene*);

delete[] array\_for\_counting\_indexes;

}

#include "nodeshape.h"

NodeShape::**NodeShape**(): QGraphicsItem()

{

//включение приема событий от обработчиков событий курсора

setAcceptHoverEvents(true);

}

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);

}

void NodeShape::***hoverEnterEvent***(QGraphicsSceneHoverEvent \*event){

NodeBrushColor = Qt::yellow;

update();

}

void NodeShape::***hoverLeaveEvent***(QGraphicsSceneHoverEvent \*event){

NodeBrushColor = Qt::green;

update();

}

3.Файл, содержащий функцию main():

**#include "widget.h"**

#include <QApplication>

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

{

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

Widget w;

w.setWindowTitle("Вертикальная печать дерева");

w.show();

return a.exec();

}

**Результат работы программы**



