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 №20

Тема: Бинарные деревья.

Вариант 15

Выполнил:

Студент группы ИВТ-20-2б

Карелов Вадим Андреевич

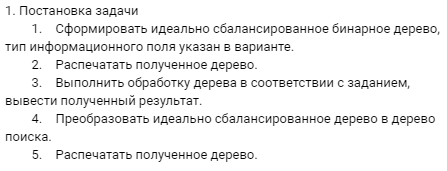
Проверила:

Доцент кафедры ИТАС

Полякова О. А.

Пермь, 2021

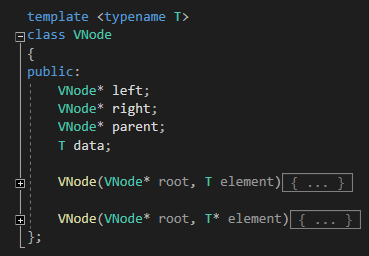
**Постановка задачи**





**Анализ задачи**

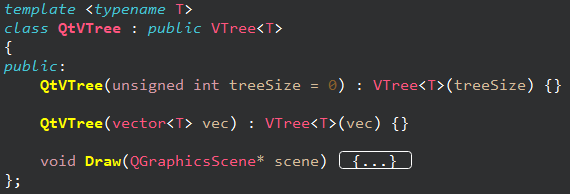
1. Используемые типы данных
   1. double
   2. Int
   3. Unsigned int
   4. VNode<T>
   5. VTree<T>
   6. Vector<T>
2. Действия над используемыми данными
   1. Int, double – арифметические операции.
3. Вид данных.
   1. Vector<T> – динамический одномерный массив.
   2. VTree<T> - бинарное дерево.
4. Структуры
   1. VNode<T> - класс, хранящий сведения об узле.



* 1. VTree<T> - класс, хранящий дерево.



* 1. QtVTree<T> - класс, наследуемый от VTree<T>. Содержит методы для отрисовки дерева в qt.



1. Организация ввода-вывода
   1. Cin
   2. Cout
   3. Отрисовка экземпляра класса QtVtree<T> в qt на виджете graphicsView.
   4. Перегрузки операторов ввода-вывода в классах. (метод ToString() в классе VTree<T>).
2. Действия для решения задачи.
   1. Для создания графического интерфейса использована IDE QtCreator.
   2. Для отрисовки написана рекурсивная функция в файле с классом QtVtree<T>.



**Код программы**

**Консольная программа**

**Main.cpp**

#include <iostream>

#include "VTree.h"

using namespace std;

int main()

{

srand(time(0));

vector<double> vec;

for (int i = 0; i < 5; i++)

{

vec.push\_back(rand() % 1000 / 10.0);

}

VTree<double> myTree(vec);

cout << myTree.ToString() << endl;

myTree.ToBalancedTree();

cout << myTree.ToString() << endl;

myTree.ToSearchTree();

cout << myTree.ToString() << endl;

cout << "MIN = " << myTree.Min() << endl;

}

**VTree.h**

#pragma once

#include <iostream>

#include <vector>

#include <string>

using namespace std;

template <typename T>

class VNode

{

public:

VNode\* left;

VNode\* right;

VNode\* parent;

T data;

VNode(VNode\* root, T element)

{

parent = root;

left = nullptr;

right = nullptr;

data = element;

}

VNode(VNode\* root, T\* element)

{

parent = root;

left = nullptr;

right = nullptr;

data = \*element;

}

};

template <typename T>

class VTree

{

private:

VNode<T>\* \_root;

int \_size;

public:

// constructors

VTree(unsigned int size = 0)

{

Delete();

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

{

Insert(0);

}

}

VTree(vector<T> vec)

{

Delete();

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

{

Insert(vec[i]);

}

}

// change tree

void Insert(T element)

{

if (\_root == nullptr)

{

\_root = new VNode<T>(nullptr, element);

\_size = 1;

}

else

{

VNode<T>\* ptr = \_root;

while (true)

{

if (element < ptr->data)

{

if (ptr->left == nullptr)

{

VNode<T>\* t = new VNode<T>(ptr, element);

ptr->left = t;

\_size++;

return;

}

else

{

ptr = ptr->left;

}

}

else

{

if (ptr->right == nullptr)

{

VNode<T>\* t = new VNode<T>(ptr, element);

ptr->right = t;

\_size++;

return;

}

else

{

ptr = ptr->right;

}

}

}

}

}

void Delete()

{

Delete\_R(\_root);

\_root = nullptr;

}

// change format

void ToBalancedTree()

{

vector<T> vec = ToVector();

Delete();

if (vec.size() > 0)

{

\_root = new VNode<T>(nullptr, vec[0]);

vec.erase(vec.begin());

\_size++;

InsertBalanced\_R(&vec, \_root, CountDeep(vec.size() + 1) - 1);

}

}

void ToSearchTree()

{

vector<T> vec = ToVector();

Delete();

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

{

Insert(vec[i]);

}

}

// get information

int Size()

{

return \_size;

}

T Min()

{

if (\_root != nullptr)

{

ToSearchTree();

VNode<T>\* ptr = \_root;

while (ptr->left != nullptr)

{

ptr = ptr->left;

}

return ptr->data;

}

return INT16\_MAX;

}

string ToString()

{

return CreateString\_R(\_root);

}

vector<T> ToVector()

{

return ToVector\_R(\_root);

}

private:

// change tree

void Delete\_R(VNode<T>\* cur)

{

if (cur != nullptr)

{

Delete\_R(cur->left);

Delete\_R(cur->right);

cur->left = nullptr;

cur->right = nullptr;

cur->parent = nullptr;

delete cur;

cur = nullptr;

\_size--;

}

}

// change format

// to balanced tree

int Sum(int n)

{

int res = 0;

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

{

res += i;

}

return res;

}

int CountDeep(int n)

{

int res = 0;

while (Sum(res) < n) res++;

return res;

}

void InsertBalanced\_R(vector<T>\* vec, VNode<T>\* parent, int deep)

{

if (deep > 0 && vec->size() > 0)

{

if (vec->size() > 0)

{

parent->left = new VNode<T>(parent, \*vec->begin());

vec->erase(vec->begin());

\_size++;

InsertBalanced\_R(vec, parent->left, deep - 1);

}

if (vec->size() > 0)

{

parent->right = new VNode<T>(parent, \*vec->begin());

vec->erase(vec->begin());

\_size++;

InsertBalanced\_R(vec, parent->right, deep - 1);

}

}

}

// get information

string CreateString\_R(VNode<T>\* cur)

{

string res;

if (cur != nullptr)

{

res += to\_string(cur->data);

res += "[";

if (cur->left != nullptr)

{

res += "l(";

res += CreateString\_R(cur->left);

res += ")";

}

if (cur->right != nullptr)

{

res += "r(";

res += CreateString\_R(cur->right);

res += ")";

}

res += "]";

res += ",";

}

return res;

}

vector<T> ToVector\_R(VNode<T>\* cur)

{

vector<T> res;

if (cur != nullptr)

{

res.push\_back(cur->data);

// take elements from left branch

vector<T> fromBranch = ToVector\_R(cur->left);

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

{

res.push\_back(fromBranch[i]);

}

// take elements from right branch

fromBranch = ToVector\_R(cur->right);

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

{

res.push\_back(fromBranch[i]);

}

}

return res;

}

};

**Код из QtCreator**

**Main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

*return* a.exec();

}

**Mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#pragma once

#include <QMainWindow>

#include "QtVTree.h"

#include "ui\_mainwindow.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*private*:

QGraphicsScene\* scene;

QtVTree<double>\* myTree;

*public*:

**MainWindow**(QWidget \*parent = *nullptr*)

{

*//* *default*

ui->setupUi(*this*);

*//* *other*

srand(time(0));

*//* *create* *scene*

scene = *new* QGraphicsScene();

scene->setSceneRect(0,0,ui->gVScene->width(), ui->gVScene->height());

ui->gVScene->setScene(scene);

scene->clear();

*//* *create* *tree*

vector<double> vec;

*for* (int i = 0; i < 6; i++)

vec.push\_back(rand() % 1000 / 10.0);

myTree = *new* QtVTree<double>(vec);

}

~***MainWindow***()

{

*//delete* *ui;*

}

*private* slots:

void **on\_pbSearch\_clicked**()

{

*if* (myTree != *nullptr*)

{

scene->clear();

myTree->ToSearchTree();

myTree->Draw(scene);

}

}

void **on\_pbBalanced\_clicked**()

{

*if* (myTree != *nullptr*)

{

scene->clear();

myTree->ToBalancedTree();

myTree->Draw(scene);

}

}

*private*:

Ui::MainWindow \*ui;

};

#endif *//* *MAINWINDOW\_H*

**QtVtree.h**

#ifndef QTVTREE\_H

#define QTVTREE\_H

#pragma once

#include <QColor>

#include <QFile>

#include <QFont>

#include <QGraphicsItem>

#include <QGraphicsScene>

#include <QPainter>

#include <QString>

#include <QtCore/qmath.h>

#include <QTextStream>

#include "VTree.h"

*template* <*typename* T>

*class* **QtVTree** : *public* VTree<T>

{

*public*:

**QtVTree**(unsigned int treeSize = 0) : VTree<T>(treeSize) {}

**QtVTree**(vector<T> vec) : VTree<T>(vec) {}

void **Draw**(QGraphicsScene\* scene)

{

*if* (QtVTree<T>::\_root != *nullptr*)

{

*//Draw\_R(QtVTree<T>::\_root,* *scene,* *scene->width()* */* *2,* *1,* *scene->width());*

Draw\_R(QtVTree<T>::\_root, scene, 0, 1, 0);

}

}

};

*//* *draw* *function*

*template* <*typename* T>

void **Draw\_R**(VNode<T>\* cur, QGraphicsScene\* scene, int centerX, int deepLevel, int oldX)

{

*//* *go* *to* *down* *elements*

*if* (cur->left != *nullptr*)

{

*//* *formula* *for* *1st* *layer* *is* *different*

int newX;

*if* (cur->parent == *nullptr*)

{

newX = -1 \* scene->width() / 2;

}

*else*

{

newX = centerX - abs(oldX - centerX) / 2;

}

Draw\_R(cur->left, scene, newX, deepLevel + 1, centerX);

}

*if* (cur->right != *nullptr*)

{

int newX;

*if* (cur->parent == *nullptr*)

{

newX = scene->width() / 2;

}

*else*

{

newX = centerX + abs(oldX - centerX) / 2;

}

Draw\_R(cur->right, scene, newX, deepLevel + 1, centerX);

}

*//* *distance* *between* *levels*

int levelSize = 60;

int startY = -1 \* scene->height() \* 5;

*//* *drawing* *line* *to* *parent*

*//* *root* *node* *dont* *need* *line* *to* *parent*

*if* (deepLevel > 1)

{

*//* *node* *coords,* *then* *parents* *coords*

QGraphicsItem \*line = scene->addLine(centerX, startY + deepLevel \* levelSize,

oldX, startY + (deepLevel - 1) \* levelSize,

QPen(Qt::*black*));

scene->addItem(line);

}

*//* *draw* *ellipse*

int r = 19;

QGraphicsEllipseItem \*ell = scene->addEllipse(centerX - r, startY + deepLevel \* levelSize - r,

r \* 2, r \* 2, QPen(Qt::*black*), QBrush(Qt::*darkYellow*));

scene->addItem(ell);

*//* *draw* *text*

QFont font("Times");

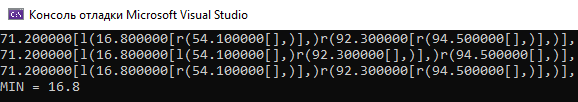
QGraphicsTextItem \*txt = scene->addText(QString::number(cur->data), font);

txt->setPos(centerX - r, startY + deepLevel \* levelSize - r);

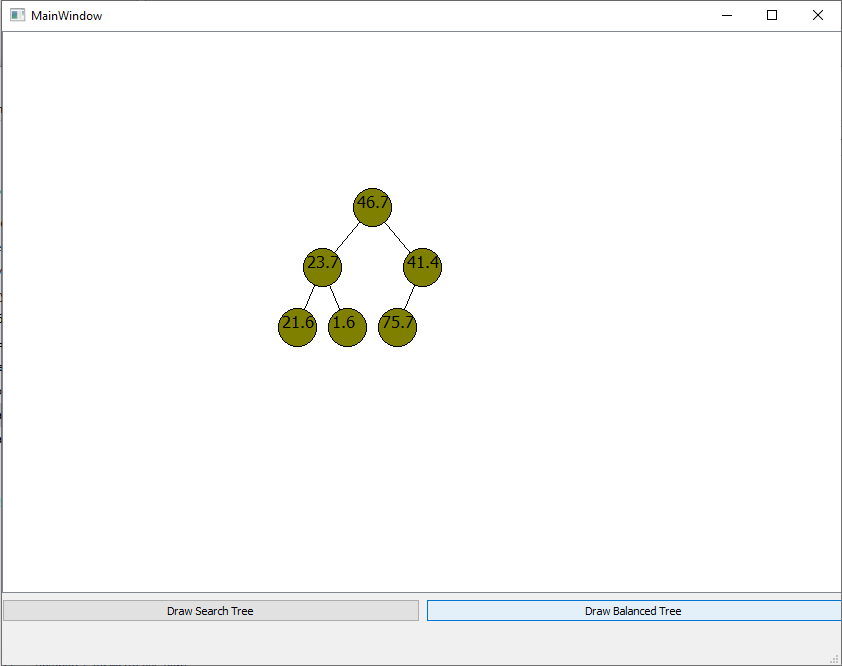
}

#endif *//* *QTVTREE\_H*

**Скриншоты**







**UML-диаграмма**

