Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе

Тема: «Деревья»

Выполнили

Студент группы РИС-22-2б

Искендеров Р.С.

Проверил доц. Кафедры ИТАС

Полякова О.А.

Пермь 2023

# Постановка задачи

1. Необходимо реализовать функции для редактирования дерева:

- Вставка узла.

- Удаление узла.

- Поиск элемента по ключу.

2. Реализовать алгоритмы обхода дерева:

2.1 Прямой

2.2 Симметричный

2.3 Обратный

3. Реализовать алгоритм балансировки дерева.

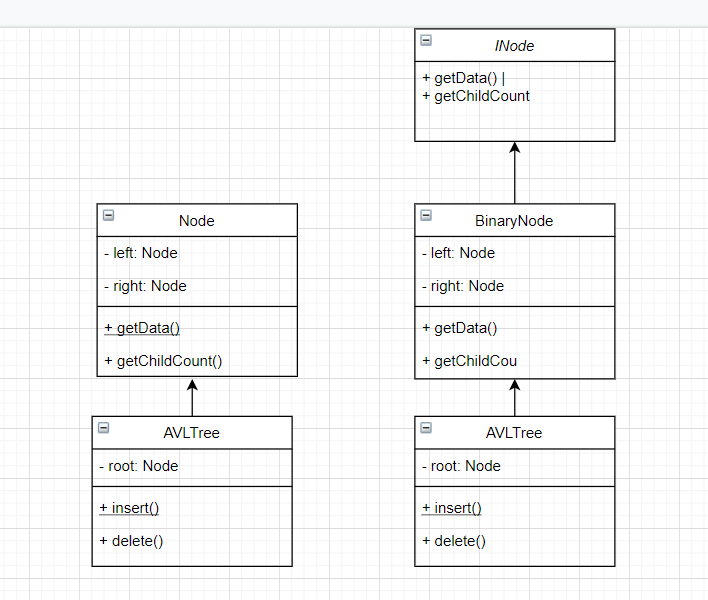
4. Реализовать вертикальную и горизонтальную печать.

5. Визуализация дерева должна быть выполнена с использованием любой доступной графической библиотеки – SFML, SDL, OpenGL и подобных.

ВАРИАНТ 25:

### Тип информационного поля double. Найти минимальный элемент в дереве.

UML – диаграмма



Код программы

**tree.h**

#ifndef TREE\_H

#define TREE\_H

#include <QTreeWidget>

class Tree {

public:

struct Node {

double data;

Node\* left;

Node\* right;

Node(double value) : data(value), left(nullptr), right(nullptr) {}

};

Tree();

~Tree();

void insert(double value);

double findMin() const;

const Node\* getRoot() const;

Node\* balanceTree();

Node\* balanceTreeHelper(std::vector<Node\*>& nodes, int start, int end);

// Методы для визуализации

QTreeWidgetItem\* createTreeItem(const Node\* node) const;

void visualize(QTreeWidget\* treeWidget) const;

private:

Node\* root;

void destroyTree(Node\* node);

Node\* insertNode(Node\* node, double value) const;

Node\* findMinNode(Node\* node) const;

int getHeight(const Node\* node) const;

int getBalanceFactor(const Node\* node) const;

Node\* rotateLeft(Node\* node);

Node\* rotateRight(Node\* node);

Node\* balanceNode(Node\* node);

};

#endif // TREE\_H

**tree.cpp**

#include "tree.h"

#include <iostream>

#include <stdexcept>

#include <map>

#include <vector>

#include <queue>

Tree::Tree() : root(nullptr) {}

Tree::~Tree() {

destroyTree(root);

}

void Tree::destroyTree(Node\* node) {

if (node != nullptr) {

destroyTree(node->left);

destroyTree(node->right);

delete node;

}

}

const Tree::Node\* Tree::getRoot() const {

return root;

}

void Tree::insert(double value) {

root = insertNode(root, value);

}

Tree::Node\* Tree::insertNode(Node\* node, double value) const {

if (node == nullptr) {

return new Node(value);

}

if (value < node->data) {

node->left = insertNode(node->left, value);

} else if (value > node->data) {

node->right = insertNode(node->right, value);

}

return node;

}

double Tree::findMin() const {

if (root == nullptr) {

throw std::runtime\_error("The tree is empty");

}

Node\* minNode = findMinNode(root);

return minNode->data;

}

Tree::Node\* Tree::findMinNode(Node\* node) const {

if (node == nullptr) {

return nullptr;

} else if (node->left == nullptr) {

return node;

}

return findMinNode(node->left);

}

//Прямой обход дерева

void preOrderTraversal(const Tree::Node\* node) {

if (node == nullptr) {

return;

}

std::cout << node->data << " "; // Вывод значения текущего узла

preOrderTraversal(node->left); // Рекурсивный обход левого поддерева

preOrderTraversal(node->right); // Рекурсивный обход правого поддерева

}

//Симметричный обход дерева

void inOrderTraversal(const Tree::Node\* node) {

if (node == nullptr) {

return;

}

inOrderTraversal(node->left); // Рекурсивный обход левого поддерева

std::cout << node->data << " "; // Вывод значения текущего узла

inOrderTraversal(node->right); // Рекурсивный обход правого поддерева

}

//Обратный обход дерева

void postOrderTraversal(const Tree::Node\* node) {

if (node == nullptr) {

return;

}

postOrderTraversal(node->left); // Рекурсивный обход левого поддерева

postOrderTraversal(node->right); // Рекурсивный обход правого поддерева

std::cout << node->data << " "; // Вывод значения текущего узла

}

//Балансировка дерева

int Tree::getHeight(const Node\* node) const {

if (node == nullptr) {

return 0;

}

int leftHeight = getHeight(node->left);

int rightHeight = getHeight(node->right);

return 1 + std::max(leftHeight, rightHeight);

}

int Tree::getBalanceFactor(const Node\* node) const {

if (node == nullptr) {

return 0;

}

int leftHeight = getHeight(node->left);

int rightHeight = getHeight(node->right);

return leftHeight - rightHeight;

}

Tree::Node\* Tree::rotateLeft(Node\* node) {

Node\* newRoot = node->right;

node->right = newRoot->left;

newRoot->left = node;

return newRoot;

}

Tree::Node\* Tree::rotateRight(Node\* node) {

Node\* newRoot = node->left;

node->left = newRoot->right;

newRoot->right = node;

return newRoot;

}

Tree::Node\* Tree::balanceNode(Node\* node) {

int balanceFactor = getBalanceFactor(node);

if (balanceFactor > 1) {

if (getBalanceFactor(node->left) < 0) {

node->left = rotateLeft(node->left);

}

return rotateRight(node);

}

if (balanceFactor < -1) {

if (getBalanceFactor(node->right) > 0) {

node->right = rotateRight(node->right);

}

return rotateLeft(node);

}

return node;

}

Tree::Node\* Tree::balanceTree() {

std::queue<Node\*> nodeQueue;

std::vector<Node\*> nodes;

if (root != nullptr) {

nodeQueue.push(root);

}

while (!nodeQueue.empty()) {

Node\* node = nodeQueue.front();

nodeQueue.pop();

nodes.push\_back(node);

if (node->left != nullptr) {

nodeQueue.push(node->left);

}

if (node->right != nullptr) {

nodeQueue.push(node->right);

}

}

std::sort(nodes.begin(), nodes.end(), [](const Node\* a, const Node\* b) {

return a->data < b->data;

});

root = balanceTreeHelper(nodes, 0, nodes.size() - 1);

return root;

}

Tree::Node\* Tree::balanceTreeHelper(std::vector<Node\*>& nodes, int start, int end) {

if (start > end) {

return nullptr;

}

int mid = start + (end - start) / 2;

Node\* node = nodes[mid];

node->left = balanceTreeHelper(nodes, start, mid - 1);

node->right = balanceTreeHelper(nodes, mid + 1, end);

return node;

}

//Вертикальная печать дерева

void printVerticalTree(const Tree::Node\* node, int level, std::map<int, std::vector<double>>& levelMap) {

if (node == nullptr) {

return;

}

levelMap[level].push\_back(node->data);

printVerticalTree(node->left, level - 1, levelMap);

printVerticalTree(node->right, level + 1, levelMap);

}

void printTreeVerticalOrder(const Tree::Node\* root) {

std::map<int, std::vector<double>> levelMap;

printVerticalTree(root, 0, levelMap);

for (const auto& levelPair : levelMap) {

std::cout << "Level " << levelPair.first << ": ";

for (const auto& value : levelPair.second) {

std::cout << value << " ";

}

std::cout << std::endl;

}

}

//Горизонтальная печать дерева

void printTreeHorizontal(const Tree::Node\* root) {

if (root == nullptr) {

return;

}

std::queue<const Tree::Node\*> nodeQueue;

nodeQueue.push(root);

while (!nodeQueue.empty()) {

int levelSize = nodeQueue.size();

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

const Tree::Node\* currentNode = nodeQueue.front();

nodeQueue.pop();

std::cout << currentNode->data << " ";

if (currentNode->left != nullptr) {

nodeQueue.push(currentNode->left);

}

if (currentNode->right != nullptr) {

nodeQueue.push(currentNode->right);

}

}

std::cout << std::endl;

}

}

**treewidget.h**

#ifndef TREEWIDGET\_H

#define TREEWIDGET\_H

#include <QTreeWidget>

#include "tree.h"

class TreeWidget : public QTreeWidget

{

Q\_OBJECT

public:

TreeWidget(QWidget \*parent = nullptr);

void setTree(const Tree::Node\* root);

private:

void clearTree();

void createTreeItem(const Tree::Node\* node, QTreeWidgetItem\* parentItem = nullptr);

};

#endif // TREEWIDGET\_H

**treewidget.cpp**

#include "treewidget.h"

#include "qlabel.h"

TreeWidget::TreeWidget(QWidget \*parent)

: QTreeWidget(parent)

{

setHeaderHidden(true);

}

void TreeWidget::setTree(const Tree::Node\* root)

{

clearTree();

createTreeItem(root);

}

void TreeWidget::clearTree()

{

clear();

}

void TreeWidget::createTreeItem(const Tree::Node\* node, QTreeWidgetItem\* parentItem)

{

if (node == nullptr)

return;

QTreeWidgetItem\* item;

if (parentItem == nullptr)

{

item = new QTreeWidgetItem(this);

setItemWidget(item, 0, new QLabel(QString::number(node->data)));

}

else

{

item = new QTreeWidgetItem(parentItem);

parentItem->addChild(item);

setItemWidget(item, 0, new QLabel(QString::number(node->data)));

}

createTreeItem(node->left, item);

createTreeItem(node->right, item);

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "treewidget.h"

#include "tree.h"

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;

TreeWidget \*treeWidget;

Tree tree;

private slots:

void insertValue();

void findMinValue();

void traversePreOrder();

void preOrderTraversal(const Tree::Node\* node, QString& result);

void traverseInOrder();

void inOrderTraversal(const Tree::Node\* node, QString& result);

void traversePostOrder();

void postOrderTraversal(const Tree::Node\* node, QString& result);

void balanceTree();

void addTreeNodes(const Tree::Node\* node, QTreeWidgetItem\* parentItem);

void printVerticalTree();

void printVerticalTreeHelper(const Tree::Node\* node, int level, std::map<int, std::vector<double>>& levelMap);

void printHorizontalTree();

};

#endif // MAINWINDOW\_H

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QLabel>

#include <QLineEdit>

#include <QPushButton>

#include <QMessageBox>

#include <QInputDialog>

#include <iostream>

#include <queue>

#include "tree.h"

#include <sstream>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

// Создание и настройка виджета дерева

treeWidget = new TreeWidget(this);

QVBoxLayout \*layout = new QVBoxLayout(ui->centralwidget);

layout->addWidget(treeWidget);

// Создание и настройка элементов интерфейса

QPushButton \*insertButton = new QPushButton("Ввод", this);

QPushButton \*findMinButton = new QPushButton("Нахождение минимального", this);

QPushButton \*preOrderButton = new QPushButton("Прямой обход", this);

QPushButton \*traverseInOrderButton = new QPushButton("Симметричный обход", this);

QPushButton \*traversePostOrderButton = new QPushButton("Обратный обход", this);

QPushButton \*balanceButton = new QPushButton("Балансировка", this);

QPushButton \*printVerticalTreeButton = new QPushButton("Вертикальная печать", this);

QPushButton \*printHorizontalTreeButton = new QPushButton("Горизонтальная печать", this);

QHBoxLayout \*buttonLayout = new QHBoxLayout;

buttonLayout->addWidget(insertButton);

buttonLayout->addWidget(findMinButton);

buttonLayout->addWidget(preOrderButton);

buttonLayout->addWidget(traverseInOrderButton);

buttonLayout->addWidget(traversePostOrderButton);

buttonLayout->addWidget(balanceButton);

buttonLayout->addWidget(printVerticalTreeButton);

buttonLayout->addWidget(printHorizontalTreeButton);

layout->addLayout(buttonLayout);

// Подключение сигналов и слотов

connect(insertButton, &QPushButton::clicked, this, &MainWindow::insertValue);

connect(findMinButton, &QPushButton::clicked, this, &MainWindow::findMinValue);

connect(preOrderButton, &QPushButton::clicked, this, &MainWindow::traversePreOrder);

connect(traverseInOrderButton, &QPushButton::clicked, this, &MainWindow::traverseInOrder);

connect(traversePostOrderButton, &QPushButton::clicked, this, &MainWindow::traversePostOrder);

connect(balanceButton, &QPushButton::clicked, this, &MainWindow::balanceTree);

connect(printVerticalTreeButton, &QPushButton::clicked, this, &MainWindow::printVerticalTree);

connect(printHorizontalTreeButton, &QPushButton::clicked, this, &MainWindow::printHorizontalTree);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::insertValue()

{

bool ok;

double value = QInputDialog::getDouble(this, "Введите значение", "Введите значение:", 0, -2147483647, 2147483647, 1, &ok);

if (ok) {

tree.insert(value);

treeWidget->setTree(tree.getRoot());

}

}

void MainWindow::findMinValue()

{

try {

double minValue = tree.findMin();

QMessageBox::information(this, "Минимальное значение", "Минимальное значение в дереве: " + QString::number(minValue));

} catch (const std::runtime\_error& e) {

QMessageBox::warning(this, "Ошибка", e.what());

}

}

void MainWindow::traversePreOrder()

{

QString result = "Прямой обход: ";

preOrderTraversal(tree.getRoot(), result);

QMessageBox::information(this, "Прямой обход", result);

}

void MainWindow::preOrderTraversal(const Tree::Node\* node, QString& result)

{

if (node == nullptr)

return;

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

preOrderTraversal(node->left, result);

preOrderTraversal(node->right, result);

}

void MainWindow::traverseInOrder()

{

QString result = "Симметричный обход: ";

inOrderTraversal(tree.getRoot(), result);

QMessageBox::information(this, "Симметричный обход", result);

}

void MainWindow::inOrderTraversal(const Tree::Node\* node, QString& result)

{

if (node == nullptr)

return;

inOrderTraversal(node->left, result);

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

inOrderTraversal(node->right, result);

}

void MainWindow::traversePostOrder()

{

QString result = "Обратный обход: ";

postOrderTraversal(tree.getRoot(), result);

QMessageBox::information(this, "Обратный обход", result);

}

void MainWindow::postOrderTraversal(const Tree::Node\* node, QString& result)

{

if (node == nullptr)

return;

postOrderTraversal(node->left, result);

postOrderTraversal(node->right, result);

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

}

void MainWindow::balanceTree()

{

// Балансировка дерева

tree.balanceTree();

// Получение обновленного корня дерева

const Tree::Node\* root = tree.getRoot();

// Очистка TreeWidget

treeWidget->clear();

// Рекурсивное добавление узлов в TreeWidget

addTreeNodes(root, nullptr);

// Раскрытие всех элементов TreeWidget

treeWidget->expandAll();

}

void MainWindow::addTreeNodes(const Tree::Node\* node, QTreeWidgetItem\* parentItem)

{

if (node == nullptr)

return;

// Создание элемента TreeWidgetItem

QTreeWidgetItem\* item = new QTreeWidgetItem();

item->setText(0, QString::number(node->data));

// Добавление элемента в TreeWidget

if (parentItem != nullptr)

parentItem->addChild(item);

else

treeWidget->addTopLevelItem(item);

// Рекурсивное добавление дочерних узлов

addTreeNodes(node->left, item);

addTreeNodes(node->right, item);

}

void MainWindow::printVerticalTreeHelper(const Tree::Node\* node, int level, std::map<int, std::vector<double>>& levelMap)

{

if (node == nullptr) {

return;

}

levelMap[level].push\_back(node->data);

printVerticalTreeHelper(node->left, level - 1, levelMap);

printVerticalTreeHelper(node->right, level + 1, levelMap);

}

void MainWindow::printVerticalTree()

{

std::map<int, std::vector<double>> levelMap;

printVerticalTreeHelper(tree.getRoot(), 0, levelMap);

// Формирование текста для вывода в окне

QString messageText;

for (const auto& levelPair : levelMap) {

messageText += "Уровень " + QString::number(levelPair.first) + ": ";

for (const auto& value : levelPair.second) {

messageText += QString::number(value) + " ";

}

messageText += "\n";

}

// Отображение всплывающего окна с вертикальным представлением дерева

QMessageBox::information(this, "Вертикальная печать дерева", messageText);

}

void MainWindow::printHorizontalTree()

{

// Создание текста для вывода в окне

QString messageText;

std::stringstream ss;

// Горизонтальная печать дерева

std::queue<const Tree::Node\*> nodeQueue;

nodeQueue.push(tree.getRoot());

while (!nodeQueue.empty()) {

int levelSize = nodeQueue.size();

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

const Tree::Node\* currentNode = nodeQueue.front();

nodeQueue.pop();

ss << currentNode->data << " ";

if (currentNode->left != nullptr) {

nodeQueue.push(currentNode->left);

}

if (currentNode->right != nullptr) {

nodeQueue.push(currentNode->right);

}

}

ss << std::endl;

}

messageText = QString::fromStdString(ss.str());

// Отображение всплывающего окна с горизонтальной печатью дерева

QMessageBox::information(this, "Горизонтальная печать дерева", messageText);

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QGraphicsScene>

#include <QGraphicsView>

#include <QGraphicsTextItem>

#include <QVector>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Результаты

