Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

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

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Бинарные деревья

Выполнил

Студент группы РИС-22-2б

Вековшинин Д. А.

Проверила

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

Постановка задачи

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

Алгоритмы:

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

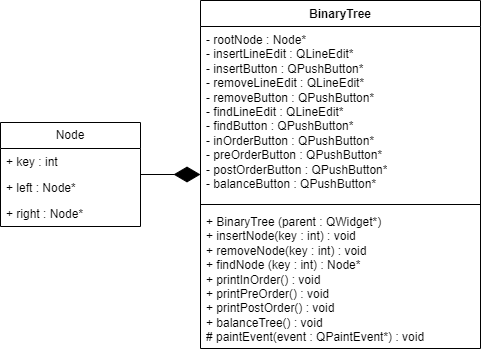
3. Реализовать алгоритм балансировки дерева.

4. Реализовать вертикальную и горизонтальную печать.

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

6. Пользовательский интерфейс по усмотрению разработчика - с условием кроссплатформенности (поощряется использование Qt или иных фреймворков).

UML-диаграмма



Код программы

#include <QtWidgets>

#include <iostream>

*struct* **Node**

{

int key;

Node\* left;

Node\* right;

};

*class* **BinaryTree** : *public* QWidget

{

Q\_OBJECT

*public*:

**BinaryTree**(QWidget\* parent = *nullptr*) : QWidget(*parent*), rootNode(*nullptr*)

{

QVBoxLayout\* layout = *new* QVBoxLayout(*this*);

*//* *Создание* *элементов* *пользовательского* *интерфейса*

QLabel\* titleLabel = *new* QLabel("", *this*);

titleLabel->setAlignment(Qt::*AlignCenter*);

layout->addWidget(*titleLabel*);

QHBoxLayout\* buttonsLayout = *new* QHBoxLayout;

insertLineEdit = *new* QLineEdit(*this*);

insertButton = *new* QPushButton("Вставить", *this*);

connect(insertButton, &QPushButton::clicked, *this*, &BinaryTree::insertButtonClicked);

buttonsLayout->addWidget(*insertLineEdit*);

buttonsLayout->addWidget(*insertButton*);

removeLineEdit = *new* QLineEdit(*this*);

removeButton = *new* QPushButton("Удалить", *this*);

connect(removeButton, &QPushButton::clicked, *this*, &BinaryTree::removeButtonClicked);

buttonsLayout->addWidget(*removeLineEdit*);

buttonsLayout->addWidget(*removeButton*);

findLineEdit = *new* QLineEdit(*this*);

findButton = *new* QPushButton("Поиск", *this*);

connect(findButton, &QPushButton::clicked, *this*, &BinaryTree::findButtonClicked);

buttonsLayout->addWidget(*findLineEdit*);

buttonsLayout->addWidget(*findButton*);

layout->addLayout(*buttonsLayout*);

QHBoxLayout\* traversalLayout = *new* QHBoxLayout;

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

connect(inOrderButton, &QPushButton::clicked, *this*, &BinaryTree::inOrderButtonClicked);

traversalLayout->addWidget(*inOrderButton*);

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

connect(preOrderButton, &QPushButton::clicked, *this*, &BinaryTree::preOrderButtonClicked);

traversalLayout->addWidget(*preOrderButton*);

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

connect(postOrderButton, &QPushButton::clicked, *this*, &BinaryTree::postOrderButtonClicked);

traversalLayout->addWidget(*postOrderButton*);

layout->addLayout(*traversalLayout*);

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

connect(balanceButton, &QPushButton::clicked, *this*, &BinaryTree::balanceTreeButtonClicked);

layout->addWidget(*balanceButton*);

setLayout(*layout*);

resize(800, 600);

}

void **insertNode**(int key)

{

insertNode(*rootNode*, key);

update();

}

void **removeNode**(int key)

{

removeNode(*rootNode*, key);

update();

}

Node\* **findNode**(int key)

{

*return* findNode(*rootNode*, key);

}

void **printInOrder**()

{

printInOrder(*rootNode*);

}

void **printPreOrder**()

{

printPreOrder(*rootNode*);

}

void **printPostOrder**()

{

printPostOrder(*rootNode*);

}

void **balanceTree**()

{

balanceTree(*rootNode*);

update();

}

*protected*:

void ***paintEvent***(QPaintEvent\* event) *override*

{

Q\_UNUSED(event);

QPainter painter(*this*);

painter.setRenderHint(QPainter::*Antialiasing*);

painter.fillRect(rect(), Qt::*white*);

*if* (rootNode)

{

int width = rect().width();

int height = rect().height();

int startX = width / 2;

int startY = 50;

drawTree(*painter*, *rootNode*, startX, startY, width / 4, height / 10);

}

}

*private*:

Node\* rootNode;

QLineEdit\* insertLineEdit;

QPushButton\* insertButton;

QLineEdit\* removeLineEdit;

QPushButton\* removeButton;

QLineEdit\* findLineEdit;

QPushButton\* findButton;

QPushButton\* inOrderButton;

QPushButton\* preOrderButton;

QPushButton\* postOrderButton;

QPushButton\* balanceButton;

void **insertNode**(Node\*& node, int key)

{

*if* (node == *nullptr*)

{

node = *new* Node{key, *nullptr*, *nullptr*};

*return*;

}

*if* (key < node->key)

insertNode(*node->left*, key);

*else* *if* (key > node->key)

insertNode(*node->right*, key);

}

Node\* **removeNode**(Node\* node, int key)

{

*if* (node == *nullptr*)

*return* *nullptr*;

*if* (key < node->key)

node->left = removeNode(*node->left*, key);

*else* *if* (key > node->key)

node->right = removeNode(*node->right*, key);

*else*

{

*if* (node->left == *nullptr* && node->right == *nullptr*)

{

*delete* node;

*return* *nullptr*;

}

*else* *if* (node->left == *nullptr*)

{

Node\* temp = node->right;

*delete* node;

*return* temp;

}

*else* *if* (node->right == *nullptr*)

{

Node\* temp = node->left;

*delete* node;

*return* temp;

}

*else*

{

Node\* minNode = findMinNode(*node->right*);

node->key = minNode->key;

node->right = removeNode(*node->right*, minNode->key);

}

}

*return* node;

}

Node\* **findNode**(Node\* node, int key)

{

*if* (node == *nullptr* || node->key == key)

*return* node;

*if* (key < node->key)

*return* findNode(*node->left*, key);

*else*

*return* findNode(*node->right*, key);

}

Node\* **findMinNode**(Node\* node)

{

*while* (node->left != *nullptr*)

node = node->left;

*return* node;

}

void **printInOrder**(Node\* node)

{

*if* (node != *nullptr*)

{

printInOrder(*node->left*);

std::cout << node->key << " ";

printInOrder(*node->right*);

}

}

void **printPreOrder**(Node\* node)

{

*if* (node != *nullptr*)

{

std::cout << node->key << " ";

printPreOrder(*node->left*);

printPreOrder(*node->right*);

}

}

void **printPostOrder**(Node\* node)

{

*if* (node != *nullptr*)

{

printPostOrder(*node->left*);

printPostOrder(*node->right*);

std::cout << node->key << " ";

}

}

int **getHeight**(Node\* node)

{

*if* (node == *nullptr*)

*return* 0;

int leftHeight = getHeight(*node->left*);

int rightHeight = getHeight(*node->right*);

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

}

void **balanceTree**(Node\*& node)

{

*if* (node == *nullptr*)

*return*;

int leftHeight = getHeight(*node->left*);

int rightHeight = getHeight(*node->right*);

*if* (leftHeight - rightHeight > 1)

{

*if* (getHeight(*node->left->left*) >= getHeight(*node->left->right*))

rotateRight(*node*);

*else*

{

rotateLeft(*node->left*);

rotateRight(*node*);

}

}

*else* *if* (rightHeight - leftHeight > 1)

{

*if* (getHeight(*node->right->right*) >= getHeight(*node->right->left*))

rotateLeft(*node*);

*else*

{

rotateRight(*node->right*);

rotateLeft(*node*);

}

}

balanceTree(*node->left*);

balanceTree(*node->right*);

}

void **rotateRight**(Node\*& node)

{

Node\* temp = node->left;

node->left = temp->right;

temp->right = node;

node = temp;

}

void **rotateLeft**(Node\*& node)

{

Node\* temp = node->right;

node->right = temp->left;

temp->left = node;

node = temp;

}

void **drawTree**(QPainter& painter, Node\* node, int x, int y, int dx, int dy)

{

*if* (node == *nullptr*)

*return*;

*if* (node->left)

{

int xLeft = x - dx;

int yLeft = y + dy;

painter.drawLine(QPoint(x, y), QPoint(xLeft, yLeft));

drawTree(*painter*, *node->left*, xLeft, yLeft, dx / 2, dy);

}

*if* (node->right)

{

int xRight = x + dx;

int yRight = y + dy;

painter.drawLine(QPoint(x, y), QPoint(xRight, yRight));

drawTree(*painter*, *node->right*, xRight, yRight, dx / 2, dy);

}

painter.setPen(Qt::*black*);

painter.setBrush(Qt::*white*);

painter.drawEllipse(QPoint(x, y), 20, 20);

painter.drawText(QRect(x - 20, y - 20, 40, 40), Qt::*AlignCenter*, QString::number(node->key));

}

*private* slots:

void **insertButtonClicked**()

{

bool ok;

int key = insertLineEdit->text().toInt(*&ok*);

*if* (ok)

{

insertNode(key);

insertLineEdit->clear();

}

}

void **balanceTreeButtonClicked**()

{

balanceTree();

update();

}

void **removeButtonClicked**()

{

bool ok;

int key = removeLineEdit->text().toInt(*&ok*);

*if* (ok)

{

removeNode(key);

removeLineEdit->clear();

}

}

void **findButtonClicked**()

{

bool ok;

int key = findLineEdit->text().toInt(*&ok*);

*if* (ok)

{

Node\* foundNode = findNode(key);

*if* (foundNode)

QMessageBox::information(*this*, "Node Found", "Node found: " + QString::number(foundNode->key));

*else*

QMessageBox::information(*this*, "Node Not Found", "Node not found");

findLineEdit->clear();

}

}

void **inOrderButtonClicked**()

{

std::cout << "In-order: ";

printInOrder();

std::cout << std::endl;

}

void **preOrderButtonClicked**()

{

std::cout << "Pre-order: ";

printPreOrder();

std::cout << std::endl;

}

void **postOrderButtonClicked**()

{

std::cout << "Post-order: ";

printPostOrder();

std::cout << std::endl;

}

};

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

{

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

BinaryTree tree;

tree.insertNode(50);

tree.insertNode(30);

tree.insertNode(70);

tree.insertNode(20);

tree.insertNode(40);

tree.insertNode(60);

tree.insertNode(80);

tree.show();

*return* app.exec();

}

#include "main.moc"

Скриншот работы программы

