Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Информатика»

Тема:Деревья.

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Протасов Н.С.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. 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).

**Описание пользовательского класса**class**BinaryTreeWidget**:publicQWidget{

public:

**BinaryTreeWidget**(QWidget\*parent=nullptr):QWidget(*parent*),root(nullptr){}

void**insertNode**(intkey){

root=insert(*root*,key);

update();

}

void**deleteNode**(intkey){

root=remove(*root*,key);

update();

}

bool**searchNode**(intkey){

returnsearch(*root*,key);

}

void**preOrderTraversal**(){

preOrder(*root*);

}

void**inOrderTraversal**(){

inOrder(*root*);

}

void**postOrderTraversal**(){

postOrder(*root*);

}

void**balanceTree**(){

root=balance(*root*);

update();

}

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

Q\_UNUSED(event);

QPainterpainter(this);

painter.setRenderHint(QPainter::Antialiasing);

intinitialX=width()/2;

intinitialY=50;

drawTree(*painter*,initialX,initialY,*root*,1);

}

private:

TreeNode\*root;

TreeNode\***insert**(TreeNode\*node,intkey){

if(node==nullptr){

returnnewTreeNode(key);

}

if(key<node->key){

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

}

elseif(key>node->key){

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

}

returnnode;

}

TreeNode\***remove**(TreeNode\*node,intkey){

if(node==nullptr){

returnnullptr;

}

if(key<node->key){

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

}

elseif(key>node->key){

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

}

else{

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

deletenode;

returnnullptr;

}

elseif(node->left==nullptr){

TreeNode\*temp=node->right;

deletenode;

returntemp;

}

elseif(node->right==nullptr){

TreeNode\*temp=node->left;

deletenode;

returntemp;

}

else{

TreeNode\*minNode=findMin(*node->right*);

node->key=minNode->key;

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

}

}

returnnode;

}

bool**search**(TreeNode\*node,intkey){

if(node==nullptr){

returnfalse;

}

if(key==node->key){

returntrue;

}

elseif(key<node->key){

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

}

else{

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

}

}

void**preOrder**(TreeNode\*node){

if(node!=nullptr){

qDebug()<<node->key;

preOrder(*node->left*);

preOrder(*node->right*);

}

}

void**inOrder**(TreeNode\*node){

if(node!=nullptr){

inOrder(*node->left*);

qDebug()<<node->key;

inOrder(*node->right*);

}

}

void**postOrder**(TreeNode\*node){

if(node!=nullptr){

postOrder(*node->left*);

postOrder(*node->right*);

qDebug()<<node->key;

}

}

TreeNode\***balance**(TreeNode\*node){

if(node==nullptr){

returnnullptr;

}

QVector<int>keys;

collectKeys(*node*,*keys*);

returnbuildBalancedTree(keys,0,keys.size()-1);

}

void**collectKeys**(TreeNode\*node,QVector<int>&keys){

if(node!=nullptr){

collectKeys(*node->left*,*keys*);

keys.append(node->key);

collectKeys(*node->right*,*keys*);

}

}

TreeNode\***buildBalancedTree**(constQVector<int>&keys,intleft,intright){

if(left>right){

returnnullptr;

}

intmiddle=(left+right)/2;

TreeNode\*node=newTreeNode(keys[middle]);

node->left=buildBalancedTree(keys,left,middle-1);

node->right=buildBalancedTree(keys,middle+1,right);

returnnode;

}

TreeNode\***findMin**(TreeNode\*node){

while(node->left!=nullptr){

node=node->left;

}

returnnode;

}

void**drawTree**(QPainter&painter,intx,inty,TreeNode\*node,intlevel){

if(node==nullptr){

return;

}

intradius=25;

intspacing=75;

intlevelHeight=80;

painter.drawEllipse(QPointF(x,y),radius,radius);

painter.drawText(QRectF(x-radius,y-radius,2\*radius,2\*radius),Qt::AlignCenter,QString::number(node->key));

if(node->left!=nullptr){

intleftX=x-spacing\*(1<<(level-1));

intleftY=y+levelHeight;

painter.drawLine(x-radius,y+radius,leftX,leftY);

drawTree(*painter*,leftX,leftY,*node->left*,level+1);

}

if(node->right!=nullptr){

intrightX=x+spacing\*(1<<(level-1));

intrightY=y+levelHeight;

painter.drawLine(x+radius,y+radius,rightX,rightY);

drawTree(*painter*,rightX,rightY,*node->right*,level+1);

}

}

};

**Функцияmain()**

intmain(intargc,char\*argv[]){

QApplicationapp(*argc*,*argv*);

BinaryTreeWidgetbinaryTreeWidget;

QPushButtoninsertButton("Insert");

QObject::connect(&insertButton,&QPushButton::clicked,[&binaryTreeWidget](){

boolok;

intkey=QInputDialog::getInt(nullptr,"Insert Node","Enter key:",0,INT\_MIN,INT\_MAX,1,*&ok*);

if(ok){

binaryTreeWidget.insertNode(key);

}

});

QPushButtondeleteButton("Delete");

QObject::connect(&deleteButton,&QPushButton::clicked,[&binaryTreeWidget](){

boolok;

intkey=QInputDialog::getInt(nullptr,"Delete Node","Enter key:",0,INT\_MIN,INT\_MAX,1,*&ok*);

if(ok){

binaryTreeWidget.deleteNode(key);

}

});

QPushButtonsearchButton("Search");

QObject::connect(&searchButton,&QPushButton::clicked,[&binaryTreeWidget](){

boolok;

intkey=QInputDialog::getInt(nullptr,"Search Node","Enter key:",0,INT\_MIN,INT\_MAX,1,*&ok*);

if(ok){

boolfound=binaryTreeWidget.searchNode(key);

QStringmessage=found?"Node found!":"Node not found!";

QMessageBox::information(nullptr,"Search Result",message);

}

});

QPushButtonpreOrderButton("Pre-order");

QObject::connect(&preOrderButton,&QPushButton::clicked,[&binaryTreeWidget](){

binaryTreeWidget.preOrderTraversal();

});

QPushButtoninOrderButton("In-order");

QObject::connect(&inOrderButton,&QPushButton::clicked,[&binaryTreeWidget](){

binaryTreeWidget.inOrderTraversal();

});

QPushButtonpostOrderButton("Post-order");

QObject::connect(&postOrderButton,&QPushButton::clicked,[&binaryTreeWidget](){

binaryTreeWidget.postOrderTraversal();

});

QPushButtonbalanceButton("Balance");

QObject::connect(&balanceButton,&QPushButton::clicked,[&binaryTreeWidget](){

binaryTreeWidget.balanceTree();

});

QVBoxLayoutlayout;

layout.addWidget(*&binaryTreeWidget*);

QHBoxLayoutbuttonLayout;

buttonLayout.addWidget(*&insertButton*);

buttonLayout.addWidget(*&deleteButton*);

buttonLayout.addWidget(*&searchButton*);

buttonLayout.addWidget(*&preOrderButton*);

buttonLayout.addWidget(*&inOrderButton*);

buttonLayout.addWidget(*&postOrderButton*);

buttonLayout.addWidget(*&balanceButton*);

QWidgetmainWidget;

mainWidget.setLayout(*&buttonLayout*);

layout.addWidget(*&mainWidget*);

QWidgetwindow;

window.setLayout(*&layout*);

window.setWindowTitle("Binary Tree");

window.setGeometry(300, 300, 500, 500);

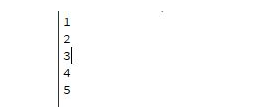
window.show();

returnapp.exec();

}

**Объяснение результатов программы**

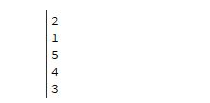
**Прямой обход**



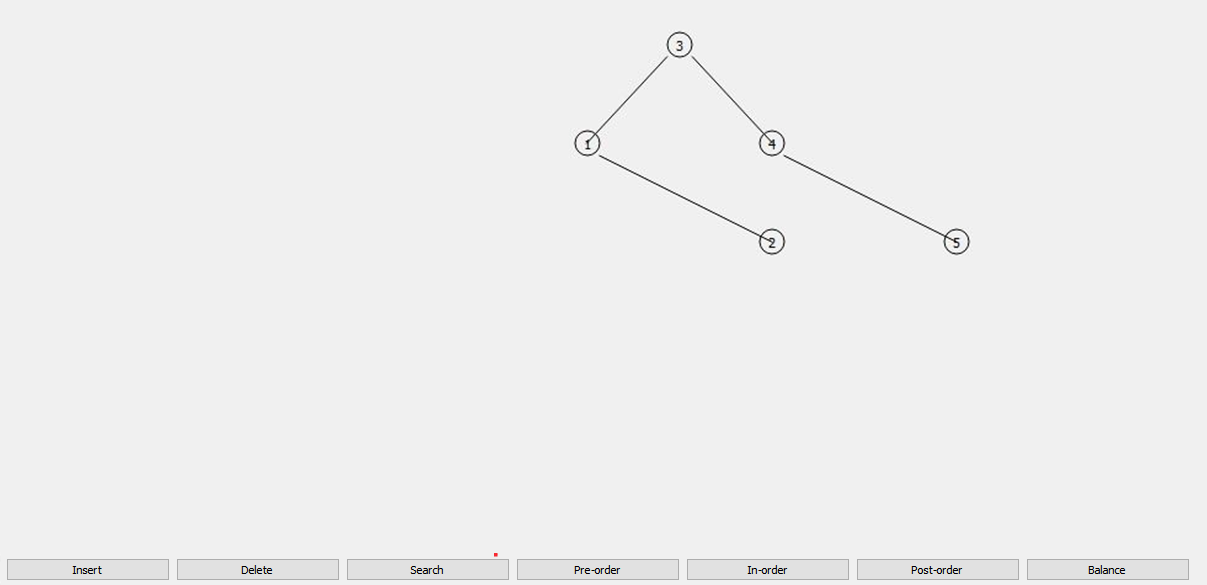
**Симметричный обход**



**Обратный обход**



**Балансировка**

****