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 №1

Тема: «Бинарные деревья»

Выполнил

Студент группы РИС-22-2б

Радостев И.А.

Проверил доц. Кафедры ИТАС

Полякова О. А.

Пермь 2023

**Постановка задачи**

1. Сформировать идеально сбалансированное бинарное дерево, тип

информационного поля указан в варианте.

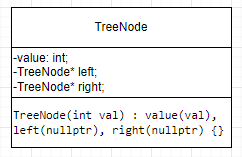
2. Распечатать полученное дерево.

3. Выполнить обработку дерева в соответствии с заданием, вывести полученный

результат.

4. Преобразовать идеально сбалансированное дерево в дерево поиска.

5. Распечатать полученное дерево.  
  
Задание: Тип информационного поля int. Найти максимальный элемент в дереве.

UML  
  
  
**Код программы**

#include <QApplication>

#include <QGraphicsScene>

#include <QGraphicsView>

#include <QGraphicsEllipseItem>

#include <QGraphicsTextItem>

#include <QBrush>

#include <iostream>

#include <cmath>

*struct* **TreeNode** {

int value;

TreeNode\* left;

TreeNode\* right;

**TreeNode**(int val) : value(val), left(*nullptr*), right(*nullptr*) {}

};

int **getTreeHeight**(TreeNode\* root) {

*if* (root == *nullptr*)

*return* 0;

int leftHeight = getTreeHeight(*root->left*);

int rightHeight = getTreeHeight(*root->right*);

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

}

void **insertNode**(TreeNode\* root, int value) {

*if* (value <= root->value) {

*if* (root->left == *nullptr*) {

root->left = *new* TreeNode(value);

} *else* {

insertNode(*root->left*, value);

}

} *else* {

*if* (root->right == *nullptr*) {

root->right = *new* TreeNode(value);

} *else* {

insertNode(*root->right*, value);

}

}

}

void **drawTree**(QGraphicsScene\* scene, QGraphicsEllipseItem\* parentEllipse, QGraphicsTextItem\* parentText, TreeNode\* node, int x, int y, int level, int levelHeight) {

*if* (node == *nullptr*) {

*return*;

}

QGraphicsEllipseItem\* ellipse = *new* QGraphicsEllipseItem(x, y, 30, 30);

QBrush brush(Qt::*white*);

ellipse->setBrush(brush);

scene->addItem(*ellipse*);

QGraphicsTextItem\* item = *new* QGraphicsTextItem(QString::number(node->value));

int textX = x + 8;

int textY = y + 5;

item->setPos(textX, textY);

scene->addItem(*item*);

*if* (parentEllipse != *nullptr*) {

int parentX = parentEllipse->rect().x() + parentEllipse->rect().width() / 2;

int parentY = parentEllipse->rect().y() + parentEllipse->rect().height();

int dx = x - parentX;

int dy = y - parentY;

int distance = std::sqrt(dx \* dx + dy \* dy);

int offsetX = (dx \* (parentEllipse->rect().width() / 2)) / distance;

int offsetY = (dy \* (parentEllipse->rect().height())) / distance;

scene->addLine(parentX + offsetX, parentY + offsetY, x + 15, y);

}

int offset = 100 \* (1 << (levelHeight - level - 1)); *//* *Увеличение* *расстояния* *между* *узлами*

drawTree(*scene*, *ellipse*, *item*, *node->left*, x - offset, y + 100, level + 1, levelHeight);

drawTree(*scene*, *ellipse*, *item*, *node->right*, x + offset, y + 100, level + 1, levelHeight);

}

TreeNode\* **createBalancedTree**(int arr[], int start, int end) {

*if* (start > end)

*return* *nullptr*;

int mid = (start + end) / 2;

TreeNode\* root = *new* TreeNode(arr[mid]);

root->left = createBalancedTree(arr, start, mid - 1);

root->right = createBalancedTree(arr, mid + 1, end);

*return* root;

}

void **printTree**(TreeNode\* root) {

*if* (root == *nullptr*)

*return*;

printTree(*root->left*);

std::cout << root->value << " ";

printTree(*root->right*);

}

int **findMaxValue**(TreeNode\* root) {

*if* (root == *nullptr*)

*return* -1; *//* *Возвращаем* *-1,* *если* *дерево* *пустое*

*while* (root->right != *nullptr*) {

root = root->right;

}

*return* root->value;

}

int main(int argc, char\*\* argv) {

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

QGraphicsScene scene;

QGraphicsView view(*&scene*);

view.show();

*//* *Формирование* *идеально* *сбалансированного* *дерева*

int arr[] = {10, 20, 30, 40, 50, 60, 70};

int arrSize = *sizeof*(arr) / *sizeof*(arr[0]);

TreeNode\* balancedRoot = createBalancedTree(arr, 0, arrSize - 1);

*//* *Определение* *высоты* *дерева*

int treeHeight = getTreeHeight(*balancedRoot*);

*//* *Визуализация* *дерева*

drawTree(*&scene*, *nullptr*, *nullptr*, *balancedRoot*, 400, 50, 0, treeHeight);

*//* *Печать* *исходного* *дерева*

std::cout << "Исходное дерево: ";

printTree(*balancedRoot*);

std::cout << std::endl;

*//* *Преобразование* *идеально* *сбалансированного* *дерева* *в* *дерево* *поиска*

*//* *Печать* *преобразованного* *дерева*

std::cout << "Преобразованное дерево: ";

printTree(*balancedRoot*);

std::cout << std::endl;

*//* *Поиск* *максимального* *элемента* *в* *дереве*

int maxValue = findMaxValue(*balancedRoot*);

std::cout << "Максимальный элемент в дереве: " << maxValue << std::endl;

*return* app.exec();

}

**Вывод программы**

