

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Пермский национальный исследовательский  
политехнический университет»**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»  
направление подготовки: 09.03.01– «Информатика и вычислительная  
техника»

**Лабораторная работа  
по дисциплине  
«Информатика»  
на тему  
«Бинарные деревья»**

Выполнил студент гр. ИВТ-23-16

Бакин Владислав Артемович

Проверил:

доц. каф. ИТАС

Полякова Ольга Андреевна

Яруллин Денис Владимирович

\_\_\_\_\_  
(оценка)

\_\_\_\_\_  
(подпись)

\_\_\_\_\_  
(дата)

г. Пермь, 2024

## 1 Вариант задания

Лабораторная работа - Бинарные деревья, вариант 5

Тип информационного поля char. Найти высоту дерева.

### 2.1 Код программы

Заголовочные файлы

```
#ifndef BINARYTREE_H
#define BINARYTREE_H
```

```
#include <iostream>
#include <cmath>
#include <vector>
#include <QtWidgets>
```

```
class Node
{
public:
    char key; // ключ. Тип данных дан
                в варианте
    Node* ptrToParent; // указатель на
                родительский узел
    Node* ptrToLeft; // указатель на
                дочерний левый узел
    Node* ptrToRight; // указатель на
                дочерний правый узел
    double x, y; // координаты того, где
                будет находится узел на сцене. Объявлен тип double, но
                по итогу при работе автоматически переводились в целые
    int posInSubTree; // вспомогательная
                нумерация узлов в строке для определения координат
    int level; // уровень узла в
                глубину

public:
    Node();
    Node(char key);
};
```

```
class BinaryTree
```

```

{
private:
    Node *ptrToTop = nullptr;           // указатель на
    корневой узел дерева

public:
    // Геттеры:
    Node* getTop();                     // возвращает указатель на
    корневой узел
    char getMax();                      // возвращает максимальное
    значение в дереве
    char getMin();                      // возвращает минимальное
    значение в дереве
    int getHeight(Node* root);          // возвращает
    высоту от узла с ключом key
    std::vector<double> getCoords(char key); // //
    возвращает координаты узла с ключом key
    std::vector<double> getCoordsOfParent(char key);
    // возвращает координаты родительского узла от узла с
    ключом key

    // Генерация дерева
    void insert(char key);              // вставка узла в базовое
    дерево поиска
    void insertToPBBT(int startIndex, int endIndex,
    std::vector<char>& roots, BinaryTree& tree); // //
    вставка узла в идеально сбалансированное
    BinaryTree regenerateToPBBT();      // чтобы
    получить из базового дерева поиска, идеально
    сбалансированное дерево

    // Проверки:
    bool isEmpty();                     // пустое ли дерево
    bool isInTree(char key);           // есть ли в дереве
    узел с ключом key
    bool hasParent(char key);           // если ли у узла с
    ключом key родительски узел

    // Обходы:
    std::vector<char> getAllElementsOfTree(); // //
    обход и добавление всех узлов в вектор
    void traverseAndAddToVector(Node* root,
    std::vector<char>& result);

```

```

        void coordCalculation(Node* root);           //
        вычисление координат для каждого узла
    };

    #endif // BINARYTREE_H
    #ifndef BINARYTREEGENERATOR_H
    #define BINARYTREEGENERATOR_H

    #include <QMainWindow>
    #include <QDebug>
    #include <QGraphicsTextItem>
    #include <vector>

    #include "binarytree.h"

    QT_BEGIN_NAMESPACE
    namespace Ui { class BinaryTreeGenerator; }
    QT_END_NAMESPACE

    class BinaryTreeGenerator : public QMainWindow
    {
        Q_OBJECT

    public:
        BinaryTreeGenerator(QWidget *parent = nullptr);
        ~BinaryTreeGenerator();

        void traverseAndPrint(Node* root, BinaryTree&
tree);           // вывод дерева

    public slots:
        void addNode();           // добавление узла
        void traverseAndPrintBase(); // сигнал вывода
        дерева поиска (базового)
        void traverseAndPrintPBBT(); // сигнал вывода
        идеально сбалансированного дерева
        void printMaxNode();       // вывод
        максимального значения в дереве
        void printMinNode();       // вывод
        минимального значения в дереве
        void printHeights();       // вывод высоты
        дерева

    private:

```

```

    BinaryTree tree;      // Дерево поиска
    BinaryTree PBBT;     // Идеально сбалансированное
    дерево

    Ui::BinaryTreeGenerator *ui;
    QGraphicsView *graphicsView;
    QGraphicsScene *scene;
};
#endif // BINARYTREEGENERATOR_H

```

**Исходные файлы**

```
#include "binarytree.h"
```

```

Node::Node()
{
    key = ' ';
    ptrToParent = nullptr;
    ptrToLeft = nullptr;
    ptrToRight = nullptr;
    x = 0.0;
    y = 0.0;
    posInSubTree = 0;
    level = 0;
}

```

```

Node::Node(char key)
{
    this->key = key;
    ptrToParent = nullptr;
    ptrToLeft = nullptr;
    ptrToRight = nullptr;
    x = 0.0;
    y = 0.0;
    posInSubTree = 0;
    level = 0;
}

```

```

Node* BinaryTree::getTop()
{
    return ptrToTop;
}

```

```
char BinaryTree::getMax()
```

```

{
    std::vector<char> allElements =
getAllElementsOfTree();
    return *max_element(allElements.begin(),
allElements.end());
}

char BinaryTree::getMin()
{
    std::vector<char> allElements =
getAllElementsOfTree();
    return *min_element(allElements.begin(),
allElements.end());
}

int BinaryTree::getHeight(Node* root)
{
    if (root == nullptr)
    {
        return 0; // Высота пустого дерева равна 0
    }
    else
    {
        // Рекурсивно находим высоту для левого и
правого поддеревьев
        int leftHeight = getHeight(root->ptrToLeft);
        int rightHeight = getHeight(root->ptrToRight);

        // Высота дерева - максимальная высота из
левого и правого поддеревьев, плюс 1 (текущий уровень)
        return std::max(leftHeight, rightHeight) + 1;
    }
}

std::vector<double> BinaryTree::getCoords(char key)
{
    std::vector<double> returnValue;
    Node *root = ptrToTop;
    while (root != nullptr)
    {
        if (root->key == key) // нашли нужный
узел
        {
            returnValue.push_back(root->x);

```

```

        returnValue.push_back(root->y);
        return returnValue;
    }
    else if (key < root->key)    // если ключ
меньше, идем в левое поддерево
    {
        root = root->ptrToLeft;
    }
    else                        // если больше, то
в правое
    {
        root = root->ptrToRight;
    }
}
return returnValue;
}

std::vector<double> BinaryTree::getCoordsOfParent(char
key)
{
    std::vector<double> returnValue;
    Node *root = ptrToTop;
    while (root != nullptr)
    {
        if (root->key == key)
        {
            returnValue.push_back(root->ptrToParent-
>x);
            returnValue.push_back(root->ptrToParent-
>y);
            return returnValue;
        }
        else if (key < root->key)
        {
            root = root->ptrToLeft;
        }
        else
        {
            root = root->ptrToRight;
        }
    }
    return returnValue;
}

```

```

void BinaryTree::insert(char key)
{
    if (ptrToTop == nullptr)    // если это первый узел
    в дереве
    {
        ptrToTop = new Node;
        ptrToTop->key = key;
        ptrToTop->ptrToLeft = nullptr;
        ptrToTop->ptrToRight = nullptr;
        ptrToTop->ptrToParent = nullptr;
        ptrToTop->x = 0.0;
        ptrToTop->y = 0.0;
        ptrToTop->posInSubTree = 1;
        ptrToTop->level = 1;
    }
    else
    {
        Node* root = ptrToTop;
        while (root != nullptr)
        {
            if (key < root->key)    // учитываем
            условие дерева поиска
            {
                if (root->ptrToLeft == nullptr)    //
                если свободное добавляем
                {
                    Node* newRoot = new Node;
                    newRoot->key = key;
                    newRoot->ptrToLeft = nullptr;
                    newRoot->ptrToRight = nullptr;
                    newRoot->ptrToParent = root;
                    newRoot->posInSubTree = root->
posInSubTree * 2 - 1;
                    newRoot->level = root->level + 1;
                    root->ptrToLeft = newRoot;

                    return;
                }
                else
                {
                    root = root->ptrToLeft;    //
                    двигаемся дальше вправо
                }
            }
        }
    }
}

```



```

        else if (key > root->key)    // учитываем
        условие дерева поиска
        {
            if (root->ptrToRight == nullptr)    //
            если свободное добавляем
            {
                Node* newRoot = new Node;
                newRoot->key = key;
                newRoot->ptrToLeft = nullptr;
                newRoot->ptrToRight = nullptr;
                newRoot->ptrToParent = root;
                if (root == ptrToTop)
                {
                    newRoot->posInSubTree = 1;
                }
                else
                {
                    newRoot->posInSubTree = root -
>posInSubTree * 2;
                }
                newRoot->level = root->level + 1;
                root->ptrToRight = newRoot;

                return;
            }
            else
            {
                root = root->ptrToRight;    //
                двигаемся дальше влево
            }
        }
    }
    else
    {
        // Значения в дереве уникальны
        std::cout << "Бинарное дерево уже имеет
узел с ключом " << key << "." << std::endl;
        std::cout << "Элемент не добавлен в
дерево." << std::endl;
        return;
    }
}
}
}
}

```

```

void BinaryTree::insertToPBBT(int startIndex, int
endIndex, std::vector<char>& roots, BinaryTree& tree)
{
    if (startIndex <= endIndex)
    {
        int midIndex = (startIndex + endIndex) / 2;

        // Вставляем узел в дерево PBBT
        tree.insert(roots[midIndex]);

        // Рекурсивно вызываем для левой и правой
        половин диапазона
        insertToPBBT(startIndex, midIndex - 1, roots,
tree);
        insertToPBBT(midIndex + 1, endIndex, roots,
tree);
    }
}

```

```

BinaryTree BinaryTree::regenerateToPBBT()
{
    // На основе базового дерева поиска создаем
    идеально сбалансированное (используя вектор всех
    элементов дерева)
    BinaryTree PBBT;
    std::vector<char> allRoots =
getAllElementsOfTree();
    sort(allRoots.begin(), allRoots.end());
    insertToPBBT(0, allRoots.size() - 1, allRoots,
PBBT);
    return PBBT;
}

```

```

bool BinaryTree::isEmpty()
{
    return ptrToTop == nullptr;
}

```

```

bool BinaryTree::isInTree(char key)
{
    Node *root = ptrToTop;
    while (root != nullptr)
    {
        if (root->key == key)

```

```

        {
            return true;
        }
        else if (key < root->key)
        {
            root = root->ptrToLeft;
        }
        else
        {
            root = root->ptrToRight;
        }
    }
    return false;
}

bool BinaryTree::hasParent(char key)
{
    if (isInTree(key) && ptrToTop->key != key)
    {
        return true;
    }
    return false;
}

std::vector<char> BinaryTree::getAllElementsOfTree()
{
    // Получаем все элементы дерева в вектор
    std::vector<char> result;
    traverseAndAddToVector(ptrToTop, result);
    return result;
}

void BinaryTree::traverseAndAddToVector(Node* root,
std::vector<char>& result)
{
    if (root != nullptr)
    {
        traverseAndAddToVector(root->ptrToLeft,
result);
        result.push_back(root->key);
        traverseAndAddToVector(root->ptrToRight,
result);
    }
}

```

```

void BinaryTree::coordCalculation(Node* root)
{
    // Алгоритм работает не идеально, но вывести дерево
    // на небольшое кол-во узлов позволяет
    int height = getHeight(ptrToTop);
    int width = 64 * height + 96 * height;
    if (root != nullptr)
    {
        coordCalculation(root->ptrToLeft);
        if (root == ptrToTop)
        {
            root->x = 0.0;
            root->y = 0.0;
        }
        else if (root->key > ptrToTop->key)
        {
            root->x = width / root->level * root-
>posInSubTree;
            root->y = (root->level - 1) * 100;
        }
        else if (root->key < ptrToTop->key)
        {
            root->x = -width + width / root->level *
root->posInSubTree;
            root->y = (root->level - 1) * 100;
        }

        qDebug() << root->key << " " << root->x << " "
<< root->y;
        coordCalculation(root->ptrToRight);
    }
}

#include "binarytreegenerator.h"
#include "ui_binarytreegenerator.h"

BinaryTreeGenerator::BinaryTreeGenerator(QWidget
*parent)
    : QMainWindow(parent)
    , ui(new Ui::BinaryTreeGenerator)
{
    ui->setupUi(this);

    graphicsView = ui->graphicsView;
    scene = new QGraphicsScene;

```

```

graphicsView->setScene(scene);

    connect(ui->addNodeBtn, &QPushButton::clicked,
this, &BinaryTreeGenerator::addNode);
    connect(ui->printTreeBtn, &QPushButton::clicked,
this, &BinaryTreeGenerator::traverseAndPrintBase);
    connect(ui->printPBBTBtn, &QPushButton::clicked,
this, &BinaryTreeGenerator::traverseAndPrintPBBT);
    connect(ui->printMaxBtn, &QPushButton::clicked,
this, &BinaryTreeGenerator::printMaxNode);
    connect(ui->printMinBtn, &QPushButton::clicked,
this, &BinaryTreeGenerator::printMinNode);
    connect(ui->printHeightsBtn, &QPushButton::clicked,
this, &BinaryTreeGenerator::printHeights);
}

BinaryTreeGenerator::~BinaryTreeGenerator()
{
    delete ui;
}

void BinaryTreeGenerator::addNode() {
    // Считываем имя узла
    QString nodeName = ui->addNodeLine->text();
    if (nodeName.isEmpty())
        return; // защита от пустого имени узла
    std::string node = nodeName.toStdString();

    tree.insert(node[0]); // добавляем узел в дерево
}

void BinaryTreeGenerator::traverseAndPrintBase()
{
    scene->clear(); // очищаем сцену
    tree.coordCalculation(tree.getTop()); //
    // вычисляем координаты
    if (tree.isEmpty())
    {
        qDebug() << "Tree is empty";
        return;
    }
    Node* root = tree.getTop();
    traverseAndPrint(root, tree); // выводим дерево
}

```

```

void BinaryTreeGenerator::traverseAndPrintPBBT()
{
    scene->clear();           // очищаем сцену
    PBBT = tree.regenerateToPBBT();           // на
основе базового дерева поиска делаем идеально
сбалансированное
    PBBT.coordCalculation(PBBT.getTop());     //
вычисляем координаты
    if (PBBT.isEmpty())
    {
        qDebug() << "PBBT is empty!";
        return;
    }
    Node* root = PBBT.getTop();
    traverseAndPrint(root, PBBT);           // выводим дерево
}

void BinaryTreeGenerator::traverseAndPrint(Node* root,
BinaryTree& tree)
{
    if (root != nullptr)
    {
        // Рекурсивно обходим левое поддереву
        traverseAndPrint(root->ptrToLeft, tree);

        // "Рисуем" узел
        QString node = QString::fromLatin1(&(root-
>key), 1);
        QGraphicsEllipseItem *ellipse = scene-
>addEllipse(root->x, root->y, 64, 64, QPen(Qt::black),
QBrush(Qt::lightGray));
        QGraphicsTextItem *textItem = scene-
>addText(node);
        textItem->setPos(ellipse-
>boundingRect().center().x() - textItem-
>boundingRect().width() / 2,
                        ellipse-
>boundingRect().center().y() - textItem-
>boundingRect().height() / 2);

        // Проверяем, не пусто ли дерево и существует
ли родитель у текущего узла
        if (!tree.isEmpty() && tree.hasParent(root-

```

```

>key) && root->ptrToParent != nullptr)
{
    // Берем координаты
    double parentX = root->ptrToParent->x;
    double parentY = root->ptrToParent->y;
    double currentNodeX = root->x;
    double currentNodeY = root->y;

    // Найдем центры эллипсов
    QPointF parentCenter(parentX + 32, parentY
+ 32);
    QPointF currentNodeCenter(currentNodeX +
32, currentNodeY + 32);

    // Создаем линию между центрами эллипсов
    scene->addLine(parentCenter.x(),
parentCenter.y(), currentNodeCenter.x(),
currentNodeCenter.y(), QPen(Qt::black));
}
ui->graphicsView->update();

// Рекурсивно обходим правое поддерево
traverseAndPrint(root->ptrToRight, tree);
}
}

void BinaryTreeGenerator::printMaxNode()
{
    char node = tree.getMax();
    // Получаем максимальный элемент дерева
    QString maxNode = QString::fromLatin1(&node, 1);
    // переводим его в QString
    ui->maxValueLabel->setText(maxNode);
    // и выводим на экран
}

void BinaryTreeGenerator::printMinNode()
{
    char node = tree.getMin();
    // Получаем минимальный элемент дерева
    QString minNode = QString::fromLatin1(&node, 1);
    // переводим его в QString
    ui->minValueLabel->setText(minNode);
    // и выводим на экран
}

```

```

}

void BinaryTreeGenerator::printHeights()
{
    int treeHeight = tree.getHeight(tree.getTop());
    // получаем высоту дерева поиска (базового)
    int PBBTHeight = PBBT.getHeight(PBBT.getTop());
    // получаем высоту идеально сбалансированного дерева
    ui->heightsValueLabel->setText("Base: " +
    QString::number(treeHeight) + "\nPBBT: " +
    QString::number(PBBTHeight)); // вывод
}
#include "binarytreegenerator.h"

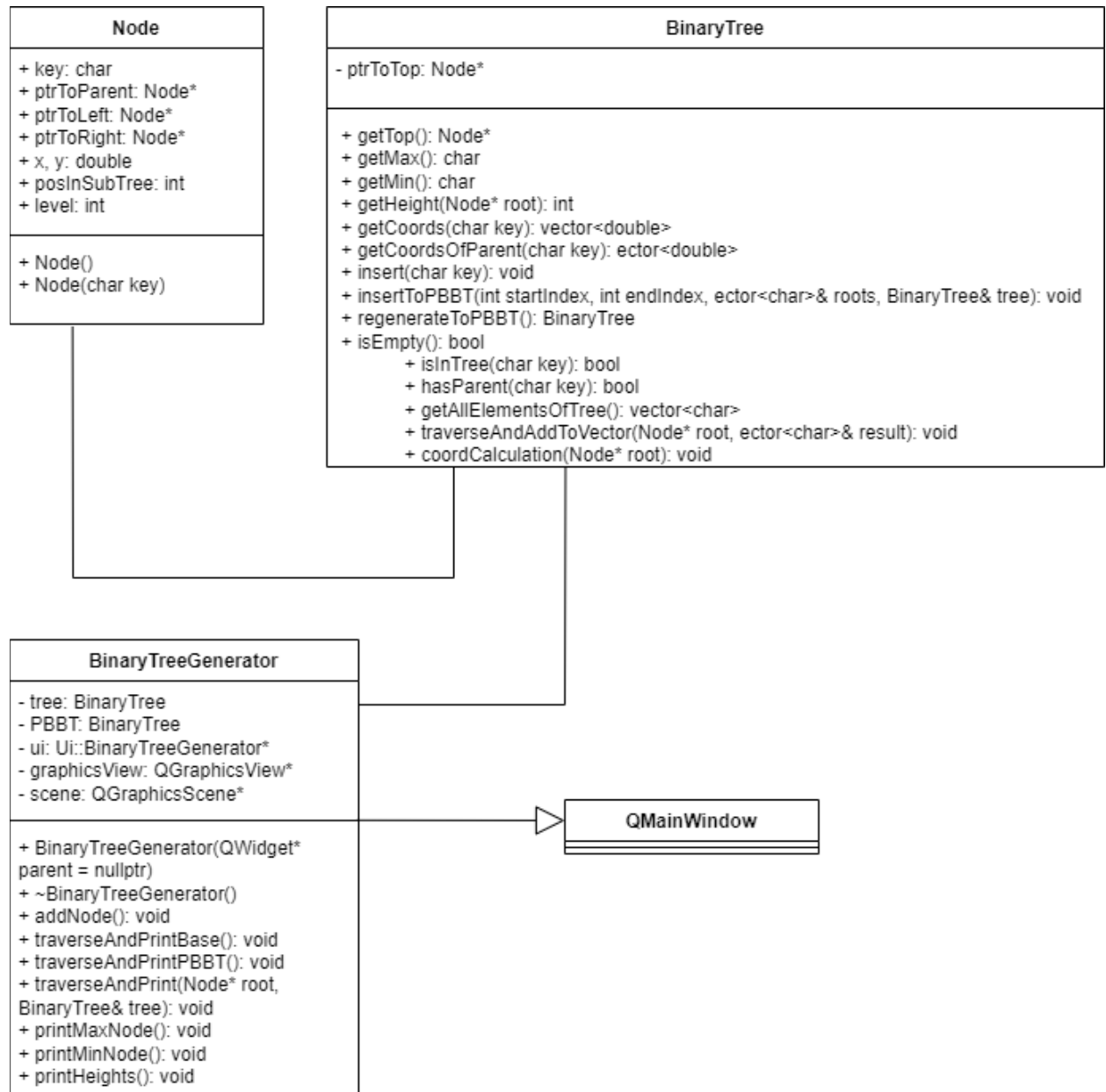
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    BinaryTreeGenerator w;
    w.resize(1920, 1080);
    w.show();
    return a.exec();
}

```



## 2.2 UML



## 3 Демонстрация работы

<https://www.youtube.com/watch?v=fY4VGkqCBEY>

[https://github.com/Meidori/PSTU\\_Labs\\_2023/assets/86147868/c13b535d-e5df-4090-a2aa-622fc825795b](https://github.com/Meidori/PSTU_Labs_2023/assets/86147868/c13b535d-e5df-4090-a2aa-622fc825795b)