МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Алгоритмы и обработка данных

Отчет по лабораторной работе № 2

Бинарные деревья поиска

Выполнили студенты группы М3О-209Б-22

Глазов В.В., Абдурахманов М.Р.

Проверила доцент, к.т.н., Дмитриева Е.А.

Москва 2023 г.

Оглавление

[Задание 3](#_Toc150764750)

[Структурные схемы алгоритмов 4](#_Toc150764751)

[**Main** 4](#_Toc150764752)

[**Menu** 5](#_Toc150764753)

[**List** 7](#_Toc150764754)

[**Append** 7](#_Toc150764755)

[**Print** 8](#_Toc150764756)

[**Clear** 10](#_Toc150764757)

[**Find** 11](#_Toc150764758)

[**DeleteById** 12](#_Toc150764759)

[Код программы 15](#_Toc150764760)

[Результат работы программы 21](#_Toc150764761)

# Задание

1. Реализовать функции вставки, поиска, удаления узла, обхода дерева, вывода дерева на экран, нахождения высоты дерева и количества узлов.
2. Реализовать дополнительно функцию: *T* – тип ключей, *D* – диапазон изменения значений ключей.

|  |  |  |  |
| --- | --- | --- | --- |
| № | *T* | *D* | Функция |
| 1 | **int** | [100; 200] | Подсчет суммы длин путей от корня до каждого из узлов, содержащих четные числа |

# Код программы

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Программирование \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Project type :Win64 Console Application \*

\*Project name :lab2\_binary\_trees.sln \*

\*File name : lab2\_binary\_trees.cpp \*

\*Language :CPP, MSVS 2022 \*

\*Programmers : Абдурахманов М.Р., Глазов В.В., М3О-209Б-22 \*

\*Modified By : \*

\*Comment : \*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//          MAIN            //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

#include <iostream>

#include <map>

#include "tree.h"

using namespace std;

//g++ main.cpp tree.cpp -o output

int main(){

    binTree tree;

    tree.insert(10);

    tree.insert(6);

    tree.insert(1);

    tree.insert(8);

    tree.insert(25);

    tree.insert(30);

    tree.insert(20);

    tree.insert(12);

    tree.insert(13);

    tree.insert(14);

    tree.insert(7);

    tree.print();

    map<int, int>\* evenLevels = tree.getEvenLevels();

    tree.printLevelMap(evenLevels);

    node\* Node = tree.preOrderSearch(30);

    if(Node) {

        cout << 30 << " in tree\n" << endl;

    } else cout << 30 << " not in tree\n" << endl;

    Node = tree.preOrderSearch(100);

    if(Node) {

        cout << 100 << " in tree\n" << endl;

    } else cout << 100 << " not in tree\n" << endl;

    bool deleted = tree.deleteNode(10);

    if(deleted) cout << "Deleted: " << 10 << endl;

    tree.print();

    deleted = tree.deleteNode(6);

    if(deleted) cout << "Deleted: " << 6 << endl;

    tree.print();

    deleted = tree.deleteNode(25);

    if(deleted) cout << "Deleted: " << 25 << endl;

    tree.print();

    char a;

    cin >> a;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//         tree.cpp         //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

#include <iostream>

#include <math.h>

#include <map>

#include "tree.h"

using namespace std;

//конструктор

binTree::binTree(){

    root = nullptr;

}

//вставка элемента

void binTree::insert(int data){

    if(!root){  //первый элемент

        root = new node{data, nullptr, nullptr};

        return;

    }

    //поиск нужного места вставки

    node \*curr = root, \*prev;

    while(curr){

        prev = curr;

        if(curr->data < data) {

            curr = curr->right;

        } else if(curr->data > data) {

            curr = curr->left;

        } else return;

    }

    if(prev->data < data) {

        prev->right = new node{data, nullptr, nullptr};

    } else if(prev->data > data) {

        prev->left = new node{data, nullptr, nullptr};

    }

}

//удаление элемента

bool binTree::deleteNode(int data){

    node \*search = preOrderSearch(data); //поиск элемента в дереве

    if(!search || !root) return false;   //элемента в дереве нет

    node \*curr = root, \*prev;   //curr - элемент, который будем удалять, prev - перед ним

    while(curr){

        prev = curr;

        if(curr->data < data){

            curr = curr->right;

        } else if(curr->data > data){

            curr = curr->left;

        }

        if(curr->data == data) break;

    }

    if(!curr->left && !curr->right){    //curr - лист дерева

        if(prev == curr) root = nullptr; //удаляем root

        if(prev->right == curr){

            prev->right = nullptr;

        } else{

            prev->left = nullptr;

        }

        delete curr;

    } else if(!curr->left && curr->right){ //справа есть ветка

        if(prev == curr) root = curr->right;    //удаляем root

        if(prev->right == curr){

            prev->right = curr->right;

        } else{

            prev->left = curr->right;

        }

        delete curr;

    } else if(curr->left && !curr->right){ //слева есть ветка

        if(prev == curr) root = curr->left;    //удаляем root

        if(prev->right == curr){

            prev->right = curr->left;

        } else{

            prev->left = curr->left;

        }

        delete curr;

    } else {    //у удаляемого элемента две ветки

        //найдем потомка слева

        node\* leftChild = curr->left, \*leftPred = curr; //левый потомок и предшествующий левого потомка

        while(leftChild->right){

            leftPred = leftChild;

            leftChild = leftChild->right;

        }

        //найдем потомка справа

        node\* rightChild = curr->right, \*rightPrev = curr;  //правый потомок и предшествующий правого потомка

        while(rightChild->left){

            rightPrev = rightChild;

            rightChild = rightChild->left;

        }

        //какой ближе к удаляемому по модулю

        if(abs(rightChild->data - curr->data) < abs(leftChild->data - curr->data)){ //если правый

            if(rightPrev->left == rightChild) rightPrev->left = rightChild->right;  //если заходили в цикл, соединим нужные ветки

            if(rightChild == curr->right) curr->right = rightChild->right;  //если не заходили в цикл(поток непосредственно справа от curr)

            curr->data = rightChild->data;  //переставили

            delete rightChild;

        } else {    //если левый

            if(leftPred->right == leftChild) leftPred->right = leftChild->left; //если заходили в цикл, соединим нужные ветки

            if(leftChild == curr->left) curr->left = leftChild->left;   //если не заходили в цикл(поток непосредственно слева от curr)

            curr->data = leftChild->data;   //переставили

            delete leftChild;

        }

    }

    return true;

}

//рекурсивный поиск начиная с узла

node\* binTree::searchFromNode(node\* Node, int data){

    //закончить есть прошли дерево или нашли

    if(!Node || Node->data == data) return Node;

    //рукурсивный поиск

    node \*left, \*right;

    left = searchFromNode(Node->left, data);

    right = searchFromNode(Node->right, data);

    //если не null

    if(left){

        return left;

    } else {

        return right;

    }

}

//preorder(прямой обход)

node\* binTree::preOrderSearch(int data){

    return searchFromNode(root, data);  //по умолчанию с корня

}

//печать дерева

void binTree::print(){

    //дерево пусто

    if(!root){

        cerr << "tree is empty\n";

        return;

    } else printf("Tree:\nNodes: %d\tHigh: %d\n", nodesQuantity(), high());

    printFromNode(root, 0); //по умолчанию с корня

    cout << "\n";

}

//рекурсивная печать начиная с узла

void binTree::printFromNode(node\* Node, int level){

    if(Node){

       if(Node->right) printFromNode(Node->right, level + 1);

        for(int i = 0; i < level; i++) cout << "   ";

        cout << Node->data << endl;

        if(Node->left) printFromNode(Node->left, level + 1);

    }

}

//количество узлов

int binTree::nodesQuantity(){

    return nodesQuantityFromNode(root);

}

//рекурсивный подсчет узлов-потомков

int binTree::nodesQuantityFromNode(node\* Node){

    if(!Node) return 0;

    return 1 + nodesQuantityFromNode(Node->left) + nodesQuantityFromNode(Node->right);

}

//высота дерева

int binTree::high(){

    return highFromNode(root) - 1;

}

//рекурсивный подсчет высоты

int binTree::highFromNode(node\* Node){

    if(!Node) return 0;

    return 1 + max(highFromNode(Node->left), highFromNode(Node->right));

}

//мапа: {четный ключ: длина пути от корня}

map<int, int>\* binTree::getEvenLevels(){

    map<int, int>\* evenLevels = new map<int, int>;

    getEvenLevelsFromNode(root, 0, \*evenLevels);

    return evenLevels;

}

//рекурсивное заполнение мапы

void binTree::getEvenLevelsFromNode(node\* Node, int level, map<int, int> &evenLevels){

    if(!Node) return;

    if(Node->data % 2 == 0) evenLevels[Node->data] = level;

    getEvenLevelsFromNode(Node->left, level+1, evenLevels);

    getEvenLevelsFromNode(Node->right, level+1, evenLevels);

}

//печать мапы

void binTree::printLevelMap(map<int, int>\* evenLevels){

    cout << "evenLevels:\n[";

    for(auto it = evenLevels->begin(); it != evenLevels->end(); it++){

        printf("{%d, %d} ", it->first, it->second);

    }

    cout << "]\n\n";

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//         tree.h           //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

#ifndef TREE\_H

#define TREE\_H

#include <map>

using namespace std;

//узел дерева

struct node{

    int data;           //ключ

    node \*left, \*right; //левый, правый узел

};

//бинарное дерево

class binTree{

    private:

        void printFromNode(node\* node , int level); //рекурсивная печать начиная с узла

        void getEvenLevelsFromNode(node\* Node, int level, map<int, int> &evenLevels);   //рекурсивное заполнение мапы

        int nodesQuantityFromNode(node\* node);      //рекурсивный подсчет узлов-потомков

        int highFromNode(node\* node);               //рекурсивный подсчет высоты

        node\* searchFromNode(node\* node, int data); //рекурсивный поиск начиная с узла

    public:

        node\* root; //корень

        binTree();  //конструктор

        void insert(int data);                          //вставка элемента

        bool deleteNode(int data);                      //удаление эелемента

        int nodesQuantity();                            //количество узлов

        int high();                                     //высота дерева

        node\* preOrderSearch(int data);                 //прямой поиск по дереву

        map<int, int>\* getEvenLevels();                 //мапа: {четный ключ: длина пути от корня}

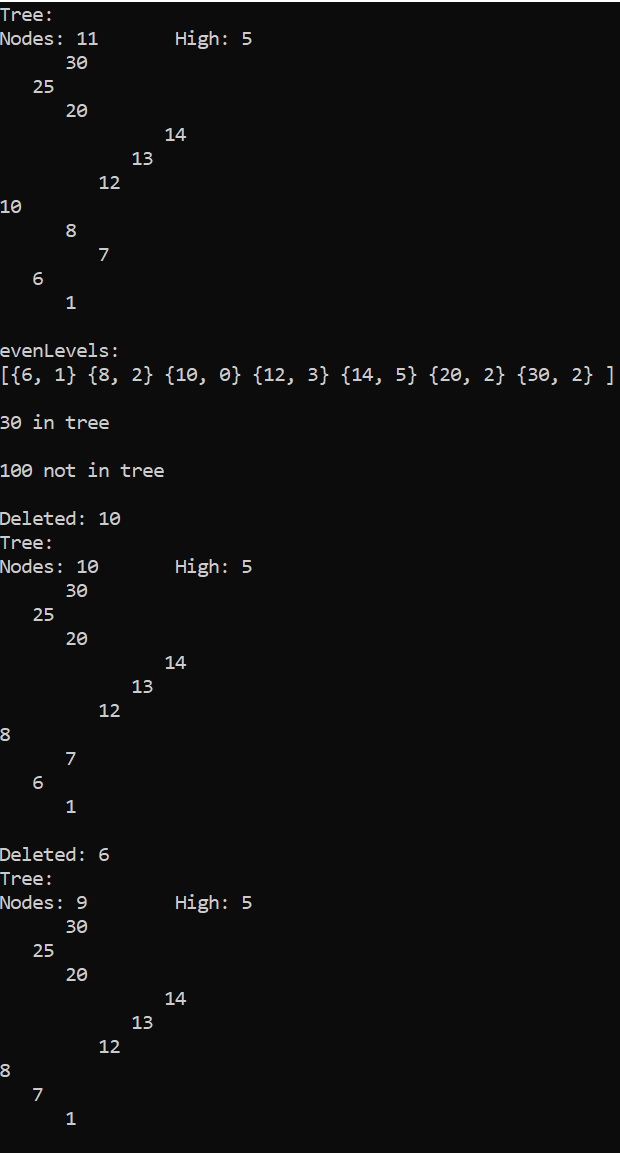
        void print();                                   //печать дерева

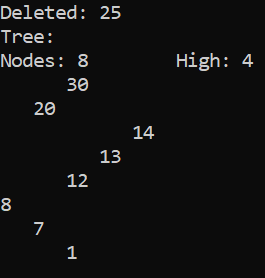
        void printLevelMap(map<int, int>\* evenLevels);  //печать мапы

};

#endif

# Результат работы программы

****

****

**Вывод**

Были изучены принципы программной реализации динамической структуры данных бинарного дерева поиска. Для этого были реализованы и использованы следующие операции:

- создание дерева

- вывод дерева на экран

- вставки элемента в дерево

- удаление узла

- обход дерева

- нахождение высоты дерева

- нахождение количества узлов

- подсчет суммы длин путей от корня до каждого из узлов, содержащих четные числа.