Институт приборостроения, автоматизации и информационных технологий

Кафедра информационных систем и цифровых технологий

Дисциплина «Алгоритмы и структуры данных»

Отчет к лабораторной работе № 4

«Бинарное поисковое дерево. Алгоритмы обхода деревьев»

Выполнил:

Василения Иван Валерьевич

Мельниченко Артём Олегович

Принял:

Рыженков Д.В.

Орёл, 2024г

Листинг файла main.cpp:

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <string>

#include "Tree.h"

using namespace std;

void readNumbersFromFile(const string& filename, Tree& tree) {

    ifstream file(filename);

    // Проверка на успешное открытие файла

    if (!file.is\_open()) {

        cerr << "Ошибка открытия файла: " << filename << endl;

    }

    string line;

    // Чтение файла построчно

    while (getline(file, line)) {

        istringstream iss(line);

        int number;

        // Чтение целых чисел из строки

        while (iss >> number) {

            tree.insert(number);

        }

    }

    file.close(); // Закрытие файла

}

int getValidNumber() {

    int number;

    while (true) {

        cout << "Введите число: ";

        cin >> number;

        if (cin.fail()) {

            cin.clear();

            cin.ignore(numeric\_limits<streamsize>::max(), '\n');

            cout << "Ошибка: введите корректное целое число!" << endl;

        }

        else {

            cin.ignore(numeric\_limits<streamsize>::max(), '\n');

            break;

        }

    }

    return number;

}

void showMenu() {

    cout << "Меню:" << endl;

    cout << "1. Добавить в дерево" << endl;

    cout << "2. Вывести дерево" << endl;

    cout << "3. Вычислить высоту" << endl;

    cout << "4. Поиск вершин, у которых количество потомков в левом поддереве отличается от количества потомков в правом поддереве на единицу" << endl;

    cout << "5. Поиск k-го в порядке слева направо листа дерева" << endl;

    cout << "6. Удаление вершины" << endl;

    cout << "0. Выход" << endl;

}

int main() {

    setlocale(LC\_ALL, "rus");

    string filename = "numbers.txt";

    Tree tree;

    readNumbersFromFile(filename, tree);

    char cmd;

    do {

        system("cls");

        showMenu();

        cin >> cmd;

        switch (cmd) {

        case '1':

            tree.insert(getValidNumber());

            break;

        case '2':

            tree.print();

            break;

        case '3':

            cout << "Высота дерева: " << tree.findHeight() << endl;

            break;

        case '4':

            tree.findUnbalancedNodes();

            break;

        case '5':

            tree.findKthLeaf(getValidNumber());

            break;

        case '6':

            tree.deleteNode(getValidNumber());

            break;

        case '0':

            cout << "Выход из программы.\n";

            return 0;

        default:

            cout << "Неверный выбор. Попробуйте еще раз.\n";

            break;

        }

        system("pause");

    } while (cmd != '0');

}

Листинг файла Tree.h:

#pragma once

#include <cstdlib>

#include <iostream>

#include <stack>

#include <utility> // Для std::pair

using namespace std;

struct Node {

    int data;

    Node\* left = nullptr;

    Node\* right = nullptr;

    Node(int value) : data(value){}

};

class Tree {

public:

    Node\* root = nullptr;

    void insert(int value) {

        root = insert(root, value);

    }

    void findUnbalancedNodes() {

        findUnbalancedNodes(root);

    }

    void findKthLeaf(int k) {

        int kValue = k;

        bool found = false;

        findKthLeaf(root, k, found);

        if (!found) {

            cout << "К-ый лист не был найден" << endl;

        }

    }

    void print() {

        print(root);

        cout << endl;

    }

    void deleteNode(int key) {

        root = deleteNode(root, key);

    }

    // Функция для нахождения высоты дерева (итеративно)

    int findHeight() {

        if (root == nullptr) return 0;

        stack<pair<Node\*, int>> stk; // Стек для хранения пар узел и уровень

        stk.push(make\_pair(root, 1)); // Начинаем с корня на уровне 1

        int maxHeight = 0;

        while (!stk.empty()) {

            pair<Node\*, int> top = stk.top();

            Node\* node = top.first;

            int level = top.second;

            stk.pop();

            if (node != nullptr) {

                maxHeight = max(maxHeight, level); // Обновляем максимальную высоту

                // Добавляем правого потомка

                stk.push(make\_pair(node->right, level + 1));

                // Добавляем левого потомка

                stk.push(make\_pair(node->left, level + 1));

            }

        }

        return maxHeight;

    }

private:

    // Вставка узла в бинарное дерево

    Node\* insert(Node\* root, int value) {

        // Если дерево пустое, создаем новый узел

        if (root == nullptr) {

            return new Node(value);

        }

        // Рекурсивный поиск подходящего места для вставки

        if (value < root->data) {

            root->left = insert(root->left, value);

        }

        else if (value > root->data) {

            root->right = insert(root->right, value);

        }

        return root; // Возврат неизмененного узла

    }

    // Рекурсивный обход для поиска вершин с разницей количества потомков == 1

    void findUnbalancedNodes(Node\* root) {

        if (root == nullptr) {

            return;

        }

        findUnbalancedNodes(root->left);

        findUnbalancedNodes(root->right);

        int leftCount = countNodes(root->left);

        int rightCount = countNodes(root->right);

        if (abs(leftCount - rightCount) == 1) {

            cout << "Вершина с данным узлом: " << root->data

                << " имеет разницу в потомках: "

                << leftCount << " (лево) и " << rightCount << " (право)" << endl;

        }

    }

    // Подсчет узлов в поддереве

    int countNodes(Node\* root) {

        if (root == nullptr) {

            return 0;

        }

        return 1 + countNodes(root->left) + countNodes(root->right);

    }

    void findKthLeaf(Node\* root, int& k, bool& found) {

        if (!root || found) {

            return;

        }

        // Сначала обрабатываем левое поддерево

        findKthLeaf(root->left, k, found);

        // Проверяем, является ли текущий узел листом

        if (!root->left && !root->right) {

            k--; // Уменьшаем k для каждого найденного листа

            if (k == 0) { // Если мы нашли k-й лист

                cout << "K-й лист: " << root->data << endl;

                found = true; // Устанавливаем флаг, что лист найден

                return;

            }

        }

        // Если не нашли в левом поддереве, продолжаем в правом

        findKthLeaf(root->right, k, found);

    }

    void print(Node\* root) {

        if (root == nullptr) { return; }

        print(root->left);

        cout << root->data << " ";

        print(root->right);

    }

    // Удаление узла со значением key из поддерева с корнем root

    Node\* deleteNode(Node\* root, int key) {

        if (!root) { return nullptr; }

        // Дальнейший поиск по поддеревьям

        if (key > root->data) {

            root->right = deleteNode(root->right, key);

        }

        else if (key < root->data) {

            root->left = deleteNode(root->left, key);

        }

        else { //key == root->data

            if (root->left == nullptr) {

                Node\* temp = root->right;

                delete root;

                return temp;

            }

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

                Node\* temp = root->left;

                delete root;

                return temp;

            }

            else { // Оба дочерних узла != nullptr

                Node\* minValueNode = root->right;

                while (minValueNode->left != nullptr) {

                    minValueNode = minValueNode->left;

                }

                // Заменяем значение удаляемого узла

                // на минимальное из правого поддереве

                root->data = minValueNode->data;

                // Удаляем из правого поддерева собственно узел с

                // минимальным значением из правого поддерева

                root->right = deleteNode(root->right, minValueNode->data);

            }

        }

        return root;

    }

};