МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФБГОУ ВО «ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра «Программная инженерия»

**Отчёт**

По лабораторной работе №3

**«Бинарное поисковое дерево. Алгоритмы обхода деревьев»**

**Выполнили:**

Студенты группы 92-ПГ

Погосян Ж.В.

**Проверили:**

Ужаринский А.Ю.

Орёл — 2021

**Код программы:**

**Main:**

#include <stdio.h>

#include <tchar.h>

#include <stack>

#include <iostream>

#include <string>

#include <fstream>

#include "Node.cpp"

int main()

{

setlocale(LC\_ALL, "russian");

int command;

char menu[256] = "1 - Поиск вершин с разным количеством потомков\n" \

"2 - Высота дерева\n" \

"3 - Поиск листа дерева слева направо \n" \

"4 - Удаление \n" \

" ";

BinarySearchTree<int> obj;

try {

if (obj.fileIn("textin.txt") == 1)

throw "Нельзя открыть файл";

}

catch (const char\* msg) {

cout << msg << endl;

}

while (1) {

obj.paintTree();

cout << endl << menu << endl;

cin >> command;

switch (command)

{

case 1:

system("cls");

cout << "Список вершин: ";

obj.count();

cout << endl << endl;

break;

case 2:

system("cls");

cout << "Высота дерева: " << obj.getHeight() << endl << endl;

break;

case 3:

{

int k;

cout << "Введите номер листа: ";

cin >> k;

system("cls");

obj.find(k);

cout << endl << endl;

}

break;

case 4:

{

int k;

cout << "Введите лист: ";

cin >> k;

system("cls");

obj.remove(k);

}

break;

default:

system("cls");

break;

}

}

obj.count();

system("pause");

return 0;

}

**Node.cpp:**

#include <stdio.h>

#include <tchar.h>

#include <stack>

#include <iostream>

#include <string>

#include <fstream>

#include <algorithm>

using namespace std;

template <typename T>

struct Node {

T key;

Node<T>\* leftNode;

Node<T>\* rightNode;

Node(T key) {

this->key = key;

leftNode = nullptr;

rightNode = nullptr;

}

};

template <typename T>

class BinarySearchTree {

private:

Node<T>\* root\_;

void addNode(Node<T>\*& node, const T& key);

Node<T>\* removeNode(Node<T>\*& node, const T& key);

Node<T>\* findMin(Node<T>\* node);

Node<T>\* findNode(const T& key) const;

void deleteTree(Node<T>\* node);

void inorderPrint(Node<T>\* node, unsigned int level);

int findVertex(Node<T>\* node, int k, int& idx);

int countOfChildren(Node<T>\* node);

unsigned int Height(const Node<T>\* node) const;

public:

BinarySearchTree();

~BinarySearchTree();

T key() const;

Node<T>\* leftNode() const;

Node<T>\* rightNode() const;

Node<T>\* root() const;

void remove(const T& value);

bool isFound(const T& value) const;

void paintTree();

unsigned int getHeight();

int fileIn(string filename);

void find(int k);

void count();

};

template<typename T>

void BinarySearchTree<T>::addNode(Node<T>\*& node, const T& key)

{

if (node) {

if (key < node->key) {

addNode(node->leftNode, key);

}

else if (key >= node->key) {

addNode(node->rightNode, key);

}

else {

return;

}

}

else {

node = new Node<T>(key);

}

}

template<typename T>

Node<T>\* BinarySearchTree<T>::removeNode(Node<T>\*& node, const T& key)

{

if (node == NULL)

return node;

if (key < node->key)

node->leftNode = removeNode(node->leftNode, key);

else if (key > node->key)

node->rightNode = removeNode(node->rightNode, key);

else {

if ((node->leftNode == NULL) || node->rightNode == NULL) {

Node<T>\* tmp = node->leftNode ? node->leftNode :

node->rightNode;

if (tmp == NULL) {

tmp = node;

node = NULL;

}

else \*node = \*tmp;

delete tmp;

}

else {

Node<T>\* tmp = findMin(node->rightNode);

node->key = tmp->key;

node->rightNode = removeNode(node->rightNode, tmp->key);

}

}

return node;

}

template<typename T>

Node<T>\* BinarySearchTree<T>::findMin(Node<T>\* node)

{

if (node->leftNode != NULL)

findMin(node->leftNode);

else return node;

}

template<typename T>

Node<T>\* BinarySearchTree<T>::findNode(const T& key) const

{

Node<T>\* currentNode = root\_;

while (currentNode) {

if (currentNode->key == key) break;

else {

if (currentNode->key < key) currentNode = currentNode->rightNode;

else currentNode = currentNode->leftNode;

}

}

return currentNode;

}

template<typename T>

void BinarySearchTree<T>::deleteTree(Node<T>\* node)

{

if (!node) return;

deleteTree(node->leftNode);

deleteTree(node->rightNode);

}

template<typename T>

int BinarySearchTree<T>::countOfChildren(Node<T>\* node)

{

Node<T>\* current = node;

if (!current) return 0;

int left = 0;

int right = 0;

left += countOfChildren(current->leftNode);

right += countOfChildren(current->rightNode);

if (abs(left - right) == 1) {

cout << current->key << " ";

}

return left + right + 1;

}

template<typename T>

int BinarySearchTree<T>::findVertex(Node<T>\* node, int k, int& idx)

{

if (!node) return 0;

findVertex(node->leftNode, k, idx);

if (idx == k && node->leftNode == NULL && node->rightNode == NULL) {

cout << "Лист:" << node->key << endl;

}

if (node->leftNode == NULL && node->rightNode == NULL) {

idx++;

}

findVertex(node->rightNode, k, idx);

}

template<typename T>

void BinarySearchTree<T>::inorderPrint(Node<T>\* node, unsigned int level)

{

if (!node) { return; }

level++;

inorderPrint(node->rightNode, level);

for (unsigned int i = 0; i < level; ++i) {

cout << ">";

}

cout << node->key << endl;

inorderPrint(node->leftNode, level);

level--;

}

template<typename T>

BinarySearchTree<T>::BinarySearchTree()

{

root\_ = nullptr;

}

template<typename T>

BinarySearchTree<T>::~BinarySearchTree()

{

deleteTree(root\_);

}

template<typename T>

T BinarySearchTree<T>::key() const

{

return root\_->key;

}

template<typename T>

unsigned int BinarySearchTree<T>::Height(const Node<T>\* node) const

{

int level = 1;

int level\_other = 0;

int max\_level = 0;

bool flag = true;

stack <Node<T>\*> steck;

stack <int> levels;

if (!node) {

return 0;

}

while (node) {

if ((node->leftNode) && (node->rightNode)) {

level++;

steck.push(node->rightNode);

levels.push(level);

node = node->leftNode;

}

else if ((node->leftNode) || (node->rightNode)) {

node = node->leftNode ? node->leftNode : node->rightNode;

level++;

}

else {

if (steck.empty())

break;

else if (steck.size() == 1 && flag) {

level\_other = level;

level = 1;

flag = false;

}

node = steck.top();

if (level > max\_level)

max\_level = level;

level = levels.top();

steck.pop();

}

}

return max(max\_level, level\_other);

}

template<typename T>

Node<T>\* BinarySearchTree<T>::leftNode() const

{

return root\_->leftNode;

}

template<typename T>

Node<T>\* BinarySearchTree<T>::rightNode() const

{

return root\_->rightNode;

}

template<typename T>

Node<T>\* BinarySearchTree<T>::root() const

{

return root\_;

}

template<typename T>

void BinarySearchTree<T>::count()

{

countOfChildren(root\_);

}

template<typename T>

void BinarySearchTree<T>::find(int k)

{

int idx = 1;

findVertex(root\_, k, idx);

}

template<typename T>

void BinarySearchTree<T>::remove(const T& key)

{

if (isFound(key)) {

removeNode(root\_, key);

return;

}

else {

cout << key << " отсутствует в дереве\n";

return;

}

}

template<typename T>

bool BinarySearchTree<T>::isFound(const T& key) const

{

Node<T>\* retNode = findNode(key);

if (retNode)

return true;

else

return false;

}

template<typename T>

void BinarySearchTree<T>::paintTree()

{

unsigned int level = 0;

inorderPrint(root\_, level);

}

template<typename T>

unsigned int BinarySearchTree<T>::getHeight()

{

return Height(root\_);

}

template<typename T>

int BinarySearchTree<T>::fileIn(string filename)

{

ifstream inFile;

inFile.open(filename);

if (!inFile) {

return 1;

}

T key;

unsigned int count;

inFile >> count;

while (count--) {

inFile >> key;

addNode(root\_, key);

}

inFile.close();

return 0;

}