

**input.txt**

18

42

38

24

41

94

37

44

75

3

15

6

24

50

71

41

90

41

81

44

**LAB4.h**

#pragma once

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

template <typename T>

class Stack {

int limit;

T\* arr;

int index = 0;

public:

Stack(int limit);

~Stack();

T pop();

void push(T& value);

void clear();

int size();

};

class Node {

private:

int value;

Node\* left = nullptr;

Node\* right = nullptr;

public:

Node(int value);

int val();

Node\*& l();

Node\*& r();

void put(int value);

bool exist(int value);

bool get\_leaf(int& result, int& counter, int k);

int show\_balanced\_count(ostream& destination);

};

class BinaryTree {

private:

Node\* first;

public:

~BinaryTree();

void put(int value);

void remove(int value);

bool exist(int value);

//Восходящий ЛПК РЕК

void show\_balanced\_count(ostream& destination);

//Нисходящий КЛП ИТЕР

int get\_height();

//Смешанный ЛКП РЕК

int get\_leaf(int k);

void show\_tree(ostream& destination);

void from\_file(ifstream& file);

};

//Последние обработанные поиском узлы

extern Node\* \_ND;

extern Node\* \_ND\_PREV;

**AISD\_LAB4.cpp**

#include "LAB4.h"

Node\* \_ND;

Node\* \_ND\_PREV;

BinaryTree tree;

int main() {

setlocale(LC\_ALL, "ru");

while (true) {

cout << "\033[0m";//Сделать текст стандартным

cout <<

"\

Список команд:\n\

1. Найти вершины, у которых количество потомков в поддеревьях отличается на единицу.\n\

2. Найти высоту дерева.\n\

3. Найти k-тый лист дерева.\n\

4. Удалить вершину.\n\

5. Отобразить дерево.\n\

\n\

6. Построение бинарного поискового дерева по данным из файла.\n\

0. Выход\n\

";

cout << "\033[1m";//Сделать текст жирным

string cmd;

cout << "Номер команды >> ";

cin >> cmd;

string answer = "";

try {

if (cmd == "1") {

tree.show\_balanced\_count(cout);

}

else if (cmd == "2") {

answer = "Высота дерева: " + to\_string(tree.get\_height());

}

else if (cmd == "3") {

int k;

cout << "Введите k >> ";

cin >> k;

answer = "Искомая вершина: " + to\_string(tree.get\_leaf(k));

}

else if (cmd == "4") {

int num;

cout << "Введите удаляемую вершину >> ";

cin >> num;

tree.remove(num);

answer = "Вершина удалена.";

}

else if (cmd == "5") {

tree.show\_tree(cout);

}

else if (cmd == "6") {

ifstream file = ifstream("input.txt", ios::in);

tree.from\_file(file);

file.close();

answer = "Дерево построено.";

}

else if (cmd == "0") {

cout << "Выход и программы.";

exit(0);

}

else {

answer = "Команда не распознана.";

}

if(answer != "")

cout << answer << endl;

cout << endl;

}

catch (exception e) {

cout << "Ошибка: " << e.what() << endl << endl;

}

}

return 0;

}

**Stack.cpp**

#include "LAB4.h"

template <typename T>

Stack<T>::Stack(int limit) {

arr = new T[limit];

this->limit = limit;

}

template <typename T>

Stack<T>::~Stack() {

delete arr;

}

template <typename T>

T Stack<T>::pop() {

if (index == 0)

throw exception("Стек пуст!");

return arr[--index];

}

template <typename T>

void Stack<T>::push(T& value) {

if (index >= limit)

throw exception("Стек переполнен!");

arr[index++] = value;

}

template <typename T>

void Stack<T>::clear() {

index = 0;

}

template <typename T>

int Stack<T>::size() {

return index;

}

**BinaryTree.cpp**

#include "Stack.cpp"

BinaryTree::~BinaryTree() {

if (!first) return;

Stack<Node\*> node\_stack(1024);

Node\* nd = nullptr;

node\_stack.push(nd);

nd = first;

while (nd) {

if (nd->l())

node\_stack.push(nd->l());

if (nd->r())

node\_stack.push(nd->r());

delete nd;

nd = node\_stack.pop();

}

}

void BinaryTree::put(int value) {

if (!first)

first = new Node(value);

else

first->put(value);

}

void BinaryTree::remove(int value) {

if (!exist(value))

throw exception("Узла с таким значением не существует!");

Node\*& link =

(\_ND\_PREV == nullptr) ? first :

(\_ND\_PREV->l() == \_ND) ? \_ND\_PREV->l() : \_ND\_PREV->r();

if (\_ND->l() && \_ND->r()) {

//ШАГ ВЛЕВО

Node\* nd\_prev = nullptr;

Node\* nd = \_ND->l();

//ВПРАВО ДО УПОРА

while (nd->r()) {

nd\_prev = nd;

nd = nd->r();

}

//Если хоть сколько-то пролистали вправо

if (nd\_prev) {

nd\_prev->r() = nd->l();

nd->l() = \_ND->l();

}

link = nd;

nd->r() = \_ND->r();

}

else if (\_ND->l())

link = \_ND->l();

else

link = \_ND->r();

delete \_ND;

}

bool BinaryTree::exist(int value) {

\_ND\_PREV = nullptr;

\_ND = nullptr;

if (!first)

throw exception("Дерево пустое!");

return first->exist(value);

}

void BinaryTree::show\_balanced\_count(ostream& destination) {

if (!first)

throw exception("Дерево пустое!");

destination << "Искомые вершины :\n";

first->show\_balanced\_count(destination);

destination << endl;

}

int BinaryTree::get\_height() {

if (!first) throw exception("Дерево пустое!");

Stack<pair<Node\*, int>> node\_stack(1024);

int result = 0;

int height = 0;

Node\* nd = first;

while (nd) {

//Обработка по условию задачи

result = max(result, height);

//Если оба потомка есть, то в стек правый

if (nd->l() && nd->r()) {

pair<Node\*, int> p = make\_pair(nd->r(), height);

node\_stack.push(p);

nd = nd->l();

height++;

}

//Если есть только правый

else if (nd->r()) {

nd = nd->r();

height++;

}

//Если есть только левый

else if (nd->l()) {

nd = nd->l();

height++;

}

//Если лист (переход к следующему узлу стека)

else {

if (node\_stack.size() == 0) break;

pair<Node\*, int> p = node\_stack.pop();

nd = p.first;

height = p.second+1;

}

}

return result;

}

int BinaryTree::get\_leaf(int k) {

if (!first)

throw exception("Дерево пустое!");

int c = 0;

int result;

if (!first->get\_leaf(result, c, k))

throw exception("Нет листа с таким номером!");

return result;

}

void BinaryTree::show\_tree(ostream& destination) {

destination << "Бинарное поисковое дерево с элементами:\n";

int h = get\_height();

Stack<Node\*> node\_stack(1024);

Node\* nd = first;

bool skip\_left = false;

if (nd)

destination << "[";

else

destination << "<Пусто>";

while (nd) {

if (nd->l() && !skip\_left) {

node\_stack.push(nd);

nd = nd->l();

destination << "Л";

}

else {

skip\_left = false;

destination << "] " << nd->val();

if (nd->r()) {

destination << ", [";

nd = nd->r();

destination << "П";

}

else {

if (node\_stack.size() == 0) break;

destination << ", [";

skip\_left = true;

nd = node\_stack.pop();

destination << "^";

}

}

}

destination << endl;

}

void BinaryTree::from\_file(ifstream& file) {

this->~BinaryTree();

first = nullptr;

int n;

while (!file.eof()) {

file >> n;

try {

put(n);

}

catch (exception e) {

cout << "Ошибка: " << e.what() << endl << endl;

}

}

}

**Node.cpp**

#include "LAB4.h"

Node::Node(int value) {

this->value = value;

}

int Node::val() {

return value;

}

Node\*& Node::l() {

return left;

}

Node\*& Node::r() {

return right;

}

void Node::put(int value) {

if (exist(value))

throw exception("Узел с таким значением уже существует!");

if (value < \_ND->value)

\_ND->left = new Node(value);

else if (value > \_ND->value)

\_ND->right = new Node(value);

}

bool Node::exist(int value) {

\_ND = this;

Node\* nd = this;

if (value < this->value) {

if (!left) return false;

nd = left;

}

else if (value > this->value) {

if (!right) return false;

nd = right;

}

else return true;

\_ND\_PREV = this;

return nd->exist(value);

}

bool Node::get\_leaf(int& result, int& counter, int k) {

if (!left && !right) {

if (k == counter) {

result = value;

return true;

}

counter++;

return false;

}

bool found = false;

if (left)

found = left->get\_leaf(result, counter, k);

if (!found && right)

found = right->get\_leaf(result, counter, k);

return found;

}

int Node::show\_balanced\_count(ostream& destination) {

int l = 0;

int r = 0;

if (left)

l = left->show\_balanced\_count(destination);

if (right)

r = right->show\_balanced\_count(destination);

if (left && right && abs(r-l) <= 1)

destination << value << " ";

return l + r + 1;

}