ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | М. А. Мурашова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «АВЛ - ДЕРЕВЬЯ ПОИСКАИ» |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134к |  |  |  | Столяров Н.С. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Цель работы**

Целью работы является изучение деревьев поиска и получение практических навыков их использования.

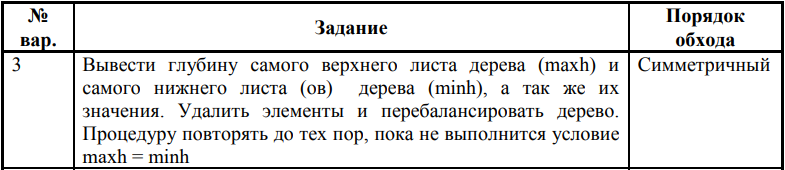
**Задание на лабораторную работу**

Разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции:

− добавлять элементы в сбалансированное дерево поиска;   
− удалять элементы из сбалансированного дерева поиска;   
− искать элементы в дереве поиска с выводом количества шагов, за которое осуществляется поиск;   
− выводить дерево на экран (любым способом доступным для восприятия);   
− выводить список, соответствующий обходу вершин, в соответствии с вариантом задания;  
− осуществлять операцию, заданную в таблице 6.

Количество элементов и порядок их ввода при создании сбалансированного дерева поиска определяется по согласованию с преподавателем.

**Вариант 3**



**Листинг**

**main.cpp**

#include <iostream>

**using** **namespace** std**;**

#include "libs/lib.h"

#include <cmath>

#include <time.h>

#include <iomanip>

// проверка ввода

#include "libs/simple\_char.h"

#include "libs/input\_validation.h"

#include "tree.h"

#define RANDOM\_MIN -100

#define RANDOM\_MAX 100

int menu**()** **{**

**while** **(true)** **{**

cout **<<** "1) Добавить элемент" **<<** endl**;**

cout **<<** "2) Добавить несколько элементов (random)" **<<** endl**;**

cout **<<** "3) Удалить элемент" **<<** endl**;**

cout **<<** "4) Поиск" **<<** endl**;**

cout **<<** "5) Обход дерева (симетрично - inOrder)" **<<** endl**;**

cout **<<** "6) Вывод глубин листов (minh, maxh)" **<<** endl**;**

cout **<<** "7) Выровнять листы (minh == maxh)" **<<** endl**;**

cout **<<** "0) Выход" **<<** endl**;**

int id **=** read\_value**(**" >>> "**,** **false,** **false,** **false);**

**if** **(**0 **<=** id **<=** 7**)** **{**

**return** id**;**

**}** **else** **{**

cout **<<** "Этого нет в меню" **<<** endl**;**

**}**

**}**

**}**

int main**()** **{**

// смена кодировки

system**(**"chcp 65001"**);**

Tree tree**;**

int menu\_i**;**

**while** **(true)** **{**

clear\_scr**();**

tree**.**show**();**

draw\_line**(**30**);**

menu\_i **=** menu**();**

**switch** **(**menu\_i**)** **{**

**case** **(**0**):**

**return** 0**;**

**break;**

**case** **(**1**):** **{**

**while** **(true)** **{**

**if** **(!**tree**.**append**(**read\_value**(**"Новый элемент: "**,** **true,** **true,** **false)))** **{**

cout **<<** "Такой элемент уже существует." **<<** endl**;**

**}** **else** **break;**

**}**

**break;**

**}**

**case** **(**2**):** **{**

double a **=** read\_value**(**"Количество новых элементов: "**,** **true,** **true,** **false);**

**for** **(**int i **=** 0**;** i **<** a**;** i**++)** **{**

tree**.**append**(**random\_int**(**RANDOM\_MIN**,** RANDOM\_MAX**));**

**}**

**break;**

**}**

**case** **(**3**):** **{**

double a **=** read\_value**(**"Удаляемый элемент: "**,** **true,** **true,** **false);**

tree**.**remove\_elem**(**a**);**

**break;**

**}**

**case** **(**4**):** **{**

double a **=** read\_value**(**"Элемент: "**,** **true,** **true,** **false);**

Node**\*** find\_element **=** tree**.**find**(**a**);**

**if** **(**find\_element **!=** **NULL)** **{**

tree**.**print\_recursion**(**""**,** find\_element**,** **false);**

**}** **else** cout **<<** "Такого элемента не существет." **<<** endl**;**

cout **<<** "Нажмите enter для продолжения..." **<<** endl**;**

getchar**();**

**break;**

**}**

**case** **(**5**):** **{**

tree**.**inOrder**(**tree**.**get\_root**());**

cout **<<** endl **<<** "Нажмите enter для продолжения..." **<<** endl**;**

getchar**();**

**break;**

**}**

**case** **(**6**):** **{**

vector**<**double**\*>** v **=** tree**.**get\_height**();**

int min\_id **=** 0**;**

int max\_id **=** 0**;**

cout **<<** "Все листы: "**;**

**for** **(**int i **=** 0**;** i **<** v**.**size**();** i**++)** **{**

cout **<<** setw**(**4**)** **<<** v**.**at**(**i**)[**0**]** **<<** ":" **<<** v**.**at**(**i**)[**1**]** **<<** " "**;**

**if** **(**v**.**at**(**i**)[**1**]** **>** v**.**at**(**min\_id**)[**1**])**

min\_id **=** i**;**

**if** **(**v**.**at**(**i**)[**1**]** **<** v**.**at**(**max\_id**)[**1**])**

max\_id **=** i**;**

**}**

cout **<<** endl**;**

cout **<<** "minh = " **<<** v**.**at**(**min\_id**)[**0**]** **<<** ":" **<<** v**.**at**(**min\_id**)[**1**]** **<<** endl**;**

cout **<<** "maxh = " **<<** v**.**at**(**max\_id**)[**0**]** **<<** ":" **<<** v**.**at**(**max\_id**)[**1**]** **<<** endl**;**

cout **<<** endl **<<** "Нажмите enter для продолжения..." **<<** endl**;**

getchar**();**

**break;**

**}**

**case** **(**7**):** **{**

vector**<**double**\*>** v**;**

int min\_id**;**

bool ok**;**

**while** **(true)** **{**

v **=** tree**.**get\_height**();**

min\_id **=** 0**;**

ok **=** **true;**

**for** **(**int i **=** 0**;** i **<** v**.**size**();** i**++)** **{**

**if** **(**v**.**at**(**i**)[**1**]** **<** v**.**at**(**min\_id**)[**1**])** **{**

ok **=** **false;**

min\_id **=** i**;**

**}**

**}**

**if** **(**ok**)** **break;**

**for** **(**int i **=** 0**;** i **<** v**.**size**();** i**++)** **{**

**if** **(**v**.**at**(**min\_id**)[**1**]** **!=** v**.**at**(**i**)[**1**])** **{**

tree**.**remove\_elem**(**v**.**at**(**i**)[**0**]);**

**}**

**}**

**}**

**break;**

**}**

**}**

**}**

**return** 0**;**

**}**

**tree.h**

#include <iostream>

**using** **namespace** std**;**

#include <vector>

// узел дерева

struct Node **{**

double elem**;** // содержимое узла

int height**;** // высота узла

// Node \*Prev; // указатель на предка

Node **\***left **=** **NULL;** // указатель на меньшего потомка

Node **\***right **=** **NULL;** // указатель на большего потомка

**};**

#include "balance.h"

class Tree **{**

public**:**

Tree**();**

bool append**(**double**);**

void remove\_elem**(**double**);**

void show**();**

Node**\*** get\_root**();**

vector**<**double**\*>** get\_height**();**

void get\_height\_recursion**(**Node**\*** ptr**,** int**);**

Node**\*** find**(**double**);**

Node**\*** balance\_node**(**Node**\*** ptr**);**

void inOrder**(**Node**\*** ptr**);**

void print\_recursion**(**string prefix**,** Node**\*** ptr**,** bool isLeft**);**

private**:**

Node **\***tree **=** **NULL;**

vector**<**double**\*>** height\_vector**;**

**};**

// конструктор

Tree**::**Tree**()** **{}**

// доавление элемента

bool Tree**::**append**(**double elem**)** **{**

bool ok **=** **true;**

**if** **(**tree **==** **NULL)** **{**

tree = new Node;

tree -> elem = elem;

tree -> height = 1;

} else {

Node \*root = tree;

int height = 1;

while (true) {

if (tree -> elem == elem) {

ok = false;

break;

} else if (tree -> elem < elem) {

if (tree -> right != NULL) {

tree = tree -> right;

} else {

tree -> right = new Node;

tree -> right -> height = height;

tree -> right -> elem = elem;

break;

}

} else {

if (tree -> left != NULL) {

tree = tree -> left;

} else {

tree -> left = new Node;

tree -> left -> height = height;

tree -> left -> elem = elem;

break;

}

}

height++;

}

tree = root;

tree = balance\_node(tree);

}

if (ok)

return true;

else

return false;

}

// удаление дерева

void Tree::remove\_elem(double elem) {

tree = remove(tree, elem);

}

// балансировка всего дерева

Node\* Tree::balance\_node(Node \*ptr) {

if (ptr -> left != NULL) {

ptr -> left = balance\_node(ptr -> left);

}

if (ptr -> right != NULL) {

ptr -> right = balance\_node(ptr -> right);

}

ptr = balance(ptr);

return ptr;

}

// вернуть указатель дерева

Node\* Tree::get\_root() {

return tree;

}

// поиск элемента

Node\* Tree::find(double elem) {

Node\* find\_elem = tree;

int steps = 0;

if (tree != NULL) {

while (true) {

steps++;

if (find\_elem -> elem < elem) {

if (find\_elem -> right != NULL) {

find\_elem = find\_elem -> right;

} else break;

} else if (find\_elem -> elem > elem) {

if (find\_elem -> left != NULL) {

find\_elem = find\_elem -> left;

} else break;

} else {

cout << "Для поисика потребовалось " << steps << " шагов." << endl;

return find\_elem;

}

}

}

return NULL;

}

// обход дерева (симетрично)

void Tree::inOrder(Node\* ptr) {

if (ptr == NULL) return;

inOrder(ptr -> left);

cout << ptr -> elem << " ";

inOrder(ptr -> right);

}

vector<double\*> Tree::get\_height() {

height\_vector.clear();

get\_height\_recursion(tree, 1);

return height\_vector;

}

// получить листы дерева

void Tree::get\_height\_recursion(Node\* ptr, int height = 1) {

if (ptr != NULL) {

if (ptr -> left == NULL && ptr -> right == NULL) {

double\* arr = new double[2];

arr[0] = ptr -> elem;

arr[1] = height;

height\_vector.push\_back(arr);

// delete arr;

// cout << ptr -> elem << " : " << height << endl;

}

else {

if (ptr -> left != NULL) get\_height\_recursion(ptr -> left, height + 1);

if (ptr -> right != NULL) get\_height\_recursion(ptr -> right, height + 1);

}

}

}

// показать дерево

void Tree::show() {

print\_recursion("", tree, false);

}

// вывод дерева (рекурсивно)

void Tree::print\_recursion(string prefix, Node\* ptr, bool isLeft) {

if (ptr != NULL) {

cout << prefix;

cout << (isLeft ? "├──" : "└──" );

cout << ptr -> elem << endl;

print\_recursion(prefix + ((isLeft) ? "│ " : " "), ptr -> right, true);

print\_recursion(prefix + ((isLeft) ? "│ " : " "), ptr -> left, false);

}

}

**balance.h**

unsigned char height**(**Node**\*** p**)** **{**

**return** p**?** p **->** height**:** 0**;**

**}**

int bfactor**(**Node**\*** p**)** **{**

**return** height**(**p **->** right**)** **-** height**(**p **->** left**);**

**}**

void fixheight**(**Node**\*** p**)** **{**

unsigned char hl **=** height**(**p**->**left**);**

unsigned char hr **=** height**(**p**->**right**);**

p **->** height **=** **(**hl **>** hr**?** hl**:** hr**)** **+** 1**;**

**}**

// поворот узла (право)

Node**\*** rotateright**(**Node**\*** p**)** **{**

Node**\*** q **=** p**->**left**;**

p**->**left **=** q**->**right**;**

q**->**right **=** p**;**

fixheight**(**p**);**

fixheight**(**q**);**

**return** q**;**

**}**

// поворот узла (лево)

Node**\*** rotateleft**(**Node**\*** q**)** **{**

Node**\*** p **=** q **->** right**;**

q **->** right **=** p **->** left**;**

p **->** left **=** q**;**

fixheight**(**q**);**

fixheight**(**p**);**

**return** p**;**

**}**

Node**\*** balance**(**Node**\*** p**)** **{**

fixheight**(**p**);**

**if** **(**bfactor**(**p**)** **==** 2**)** **{**

**if** **(**bfactor**(**p **->** right**)** **<** 0**)**

p**->**right **=** rotateright**(**p **->** right**);**

**return** rotateleft**(**p**);**

**}**

**if** **(**bfactor**(**p**)** **==** **-**2**)** **{**

**if** **(**bfactor**(**p **->** left**)** **>** 0**)**

p **->** left **=** rotateleft**(**p **->** left**);**

**return** rotateright**(**p**);**

}

return p;

}

////////////////

Node \*findmin(Node \*p) {

return p->left? findmin(p->left): p;

}

Node\* removemin(Node\* p) {

if( p->left==0 )

return p->right;

p->left = removemin(p->left);

return balance(p);

}

Node\* remove(Node\* p, int k) {

if( !p ) return 0;

if( k < p->elem )

p->left = remove(p->left,k);

else if( k > p->elem )

p->right = remove(p->right,k);

else {

Node\* q = p->left;

Node\* r = p->right;

delete p;

if( !r ) return q;

Node\* min = findmin(r);

min->right = removemin(r);

min->left = q;

return balance(min);

}

return balance(p);

}

**Результат работы**

****

**Вывод**

Мы изучили деревья поиска и получили практические навыки их использования.