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 ПОИСКА» |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

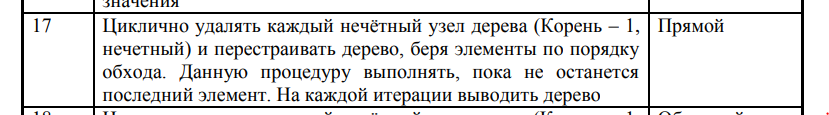
РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134к |  |  |  | Костяков Н.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

* 1. **Цель работы** Целью работы является изучение деревьев поиска и получение практических навыков их использования.
  2. **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**



**Листинг**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

struct node // структура для представления узлов дерева

{

int key;

unsigned char height;

node\* left= nullptr;

node\* right = nullptr;

node\* top = this;

node(int k) { key = k; left = right = 0; height = 1; }

};

node\* find(node\* tree, int v) {

node\* cur = tree;

while (cur)

{

if (cur->key < v)

{

cur = cur->right;

}

else if (cur->key > v) {

cur = cur->left;

}

if (cur and cur->key == v)

{

return cur;

}

}

return 0;

}

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;

}

node\* rotateright(node\* p) // правый поворот вокруг p

{

node\* q = p->left;

p->left = q->right;

q->right = p;

fixheight(p);

fixheight(q);

return q;

}

node\* rotateleft(node\* q) // левый поворот вокруг q

{

node\* p = q->right;

q->right = p->left;

p->left = q;

fixheight(q);

fixheight(p);

return p;

}

node\* balance(node\* p) // балансировка узла 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; // балансировка не нужна

}

node\* insert(node\* p, int k) // вставка ключа k в дерево с корнем p

{

if (!p) return new node(k);

if (k < p->key)

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

else

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

return balance(p);

}

node\* findmin(node\* p) // поиск узла с минимальным ключом в дереве p

{

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

}

node\* removemin(node\* p) // удаление узла с минимальным ключом из дерева p

{

if (p->left == 0)

return p->right;

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

return balance(p);

}

node\* remove(node\* p, int k) // удаление ключа k из дерева p

{

if (!p) return 0;

if (k < p->key)

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

else if (k > p->key)

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

else // k == p->key

{

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);

}

void add(node \*root, int key) {

node\* current = root;

node \*top = current->top;

bool flag = 1;

while (current)

{

if (current->key>key )

{

if (current->left != nullptr)

{

insert(current, key);

break;

}

current = current->left;

}

if (current->key < key )

{

if (current->right != nullptr)

{

insert(current, key);

break;

}

current = current->right;

}

}

root->top = top;

}

void show(node\* root, int space = 0) {

if (!root)

return;

enum { COUNT = 2 };

space += COUNT;

show(root->right, space);

for (int i = COUNT; i < space; ++i)

std::cout << " ";

std::cout << root->key << std::endl;

show(root->left, space);

}

void serch(node\* tree, int key) {

node\* current = tree;

int i = 0;

while (i<=tree->height)

{

i++;

if (current==nullptr)

{

return;

}

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

if (current->key!=key)

{

if (key>current->key)

{

current = current->right;

}

else if (key < current->key)

{

current = current->left;

}

}

else

{

break;

}

}

}

struct arr {

int value;

arr\* next = nullptr;

arr(int v = NULL) { value = v; }

};

void arr\_add(arr\* a, int v) {

while (a->next)

{

a = a->next;

}

a->next = new arr(v);

}

void walk(node\* tree, arr\* A) {

if (tree)

{

cout << tree->key<<" ";

arr\_add(A, tree->key);

walk(tree->left, A);

walk(tree->right, A);

}

}

void even\_del(node\* tree, arr\* num) {

if (tree->left==nullptr)

{

int temp = tree->key;

tree->key = tree->right->key;

tree->right->key = temp;

remove(tree, tree->right->key);

cout << tree->key << "\n";

return;

}

if (tree)

{

int i = 1;

for (arr\* current = num; current; current = current->next) {

if (i%2==1)

{

remove(tree, current->value);

}

i++;

}

}

}

void task(node\* root) {

arr A;

show(root);

walk(root, &A);

while (root->height > 2)

{

A = NULL;

cout << "\n/////////////////////////////////////////////////////////\n\n";

walk(root, &A);

even\_del(root, &A);

cout << "\n";

show(root);

cout << "/////////////////////////////////////////////////////////\n\n";

}

remove(root, root->left->key);

int temp = root->key;

root->key = root->right->key;

root->right->key = temp;

remove(root, root->right->key);

cout << root->key;

}

int main()

{

srand(time(0));

node\* root = new node(rand()%60+20);

for (int i = 0; i < 26; i++) {

insert(root, rand() % 100);

}

int v = 10;

cout << "1 - to show\n2 - to add\n3 - to delete\n4 - to search\n5 - to direct walk\n6 - to delete even\n\n";

int key;

arr A;

while (v!=0)

{

cin >> v;

switch (v)

{

case 1:

show(root);

cout << "\n";

break;

case 2:

cout << "enter key: ";

cin >> key;

if (find(root, key) != 0)

{

cout << "Enter uniq key\n";

break;

}

insert(root, key);

cout << "Added\n";

break;

case 3:

cout << "enter key: ";

cin >> key;

remove(root, key);

cout << "Deleted";

cout << "\n";

break;

case 4:

cout << "enter key: ";

cin >> key;

serch(root, key);

cout << "\n";

break;

case 5:

walk(root, &A);

cout << "\n";

break;

case 6:

task(root);

root = new node(rand() % 60 + 20);

cout << "\n";

break;

default:

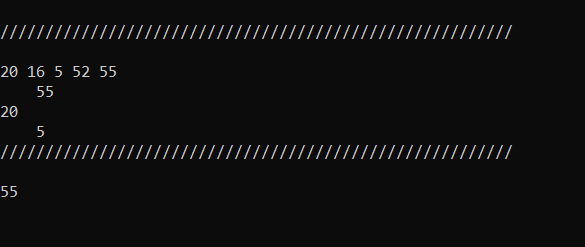
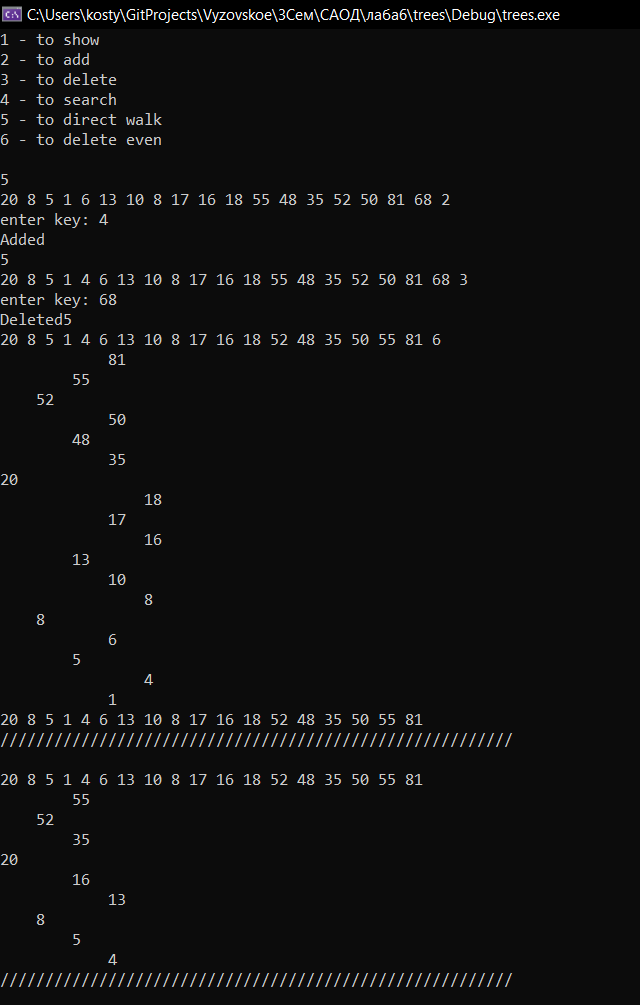
break;

}

}

}

**Результат работы программы**



**Вывод**

**Я изучил структуру деревьев и написал свое собственное**