ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

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

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4131 |  |  |  | В.А. Алексеев |
|  |  |  | подпись, дата |  | инициалы, фамилия |

* 1. Цель работы

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

* 1. 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 обратному обходу вершин;

− поиск максимальной глубины листа

− цикличное добавление элементов в дерево, пока высота всех листов дерева не будет одинакова, но не превысит изначально найденного максимального значения

* 1. Листинг

#include<iostream>

using namespace std;

struct tree {

int data;

int height;

int minheight;

//tree\* pred;

tree\* left;

tree\* right;

};

void reheight(tree\*);

tree\* balance(tree\* p);

int tab = 0;

tree\* add(tree\*& curr, int curdata) {

if (curr == NULL) {

tree\* c = new tree;

c->data = curdata;

c->height = 0;

c->minheight = 0;

c->left = NULL;

c->right = NULL;

curr = c;

return curr;

}

else {

if (curdata > curr->data) {

curr->right = balance(add(curr->right, curdata));

}

else {

curr->left = balance(add(curr->left, curdata));

}

}

reheight(curr);

return balance(curr);

}

// правый поворот вокруг q

tree\* left\_rotate(tree\* a) {

tree\* q = a->right->left;

tree\* b = a->right;

b->left = a;

a->right = q;

reheight(a);

reheight(b);

return b;

}

// левый поворот вокруг p

tree\* right\_rotate(tree\* a) {

tree\* q = a->left->right;

tree\* b = a->left;

b->right = a;

a->left = q;

reheight(a);

reheight(b);

return b;

}

void print(tree\* root) {

if (root == NULL)

return;

tab += 5;

print(root->right

for (int i = 0; i < tab; i++)

cout << " ";

cout << root->data << endl;

print(root->left);

tab -= 5;

return;

}

int height(tree\* s) {

if (s != NULL)

return s->height;

return -1;

}

int minheight(tree\* s) {

if (s != NULL)

return s->minheight;

return 0;

}

void reminheight(tree\* R) {

int hl = minheight(R->left);

int hr = minheight(R->right);

if (R->left == NULL || R->right == NULL) R->minheight = 0;

else if (hl < hr) R->minheight = hl + 1;

else R->minheight = hr + 1;

}

void reheight(tree\* R) {

reminheight(R);

int hl = height(R->left);

int hr = height(R->right);

if (hl > hr) R->height = hl + 1;

else R->height = hr + 1;

}

void fill(tree\* root) {

if (root == NULL) return;

while (root->height != root->minheight) {

if (root->right == NULL) {

root->right = add(root->right, root->data + 1);

}

else if (root->right->minheight == root->minheight - 1) {

if (root->right->height - root->right->minheight > 0)

fill(root->right);

else {

root->right->left = add(root->right->left, root->right->data - 1);

root->right->right = add(root->right->right, root->right->data + 1);

reheight(root->right);

}

}

if (root->left == NULL) {

root->left = add(root->left, root->data - 1);

}

else if (root->left->minheight == root->minheight - 1) {

if (root->left->height - root->left->minheight > 0)

fill(root->left);

else {

root->left->left = add(root->left->left, root->left->data - 1);

root->left->right = add(root->left->right, root->left->data + 1);

reheight(root->left);

}

}

reheight(root);

}

reheight(root);

return;

}

int nodebalance(tree\* p)

{

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

}

tree\* balance(tree\* p)

{

if (p == NULL)

return NULL;

if (nodebalance(p) == 2)

{

if (nodebalance(p->right) < 0)

p->right = right\_rotate(p->right);

return left\_rotate(p);

}

if (nodebalance(p) == -2)

{

if (nodebalance(p->left) > 0)

p->left = left\_rotate(p->left);

return right\_rotate(p);

}

reheight(p);

return p;

}

void order(tree\* root) {

if (root == NULL) return;

order(root->left);

order(root->right);

cout << " " << root->data << endl;

}

int step = 0;

tree\* found(tree\* root, int n) {

if (root == NULL) {

cout << "Такой элемент не найден\n";

step = 0;

return NULL;

}

step++;

if (n == root->data) {

step--;

cout << "Шагов поиска : " << step << endl;

step = 0;

return root;

}

else if (n > root->data)

found(root->right, n);

else

found(root->left, n);

}

tree\* foundmin(tree\* root) {

if (root->left == NULL)

return root;

else

foundmin(root->left);

}

tree\* deletmin(tree\* R) {

if (R->left == NULL) {

return R->right;

}

R->left = deletmin(R->left);

return balance(R);

}

tree\* del(tree\* root, int val) {

if (root == NULL) return 0;

if (val < root->data)

root->left = del(root->left, val);

else if (val > root->data)

root->right = del(root->right, val);

else {

tree\* l = root->left;

tree\* r = root->right;

delete root;

if (r == NULL) return balance(l);

tree\* min = foundmin(r);

min->right = deletmin(r);

min->left = l;

reheight(min);

return balance(min);

}

return balance(root);

}

void main() {

setlocale(LC\_ALL, "rus");

tree\* root = new tree;

cout << "Создание первого элемента дерева.\nЗначение : ";

cin >> root->data;

root->height = 0;

root->minheight = 0;

root->left = NULL;

root->right = NULL;

//root->pred = NULL;

int k = 5;

while (k != 0) {

switch (k)

{

case 1: {

cout << "значение : ";

int k;

cin >> k;

found(root, k);

break;

}case 2: {

cout << "Добавление элемента.\nВведите значение : ";

int m;

cin >> m;

root = add(root, m);

break;

}case 3: {

print(root);

break;

}case 4: {

cout << "Обход : \n";

order(root);

break;

}case 5: {

cout << "Меню\n";

cout << "1 - Поиск.\n";

cout << "2 - Добавление элемента.\n";

cout << "3 - Показать дерево.\n";

cout << "4 - Обход.\n";

cout << "5 - Меню.\n";

cout << "6 - Максимальная глубина листа\n";

cout << "7 - Удаление.\n";

cout << "8 - Заполнение.\n";

cout << "0 - выход.\n";

break;

}case 6: {

cout << "Максимальная глубина листа : ";

reheight(root);

cout << root->height << endl;

break;

}case 7: {

int m;

cout << "Какой удалить : ";

cin >> m;

root = del(root, m);

reheight(root);

break;

}case 8: {

fill(root);

break;

}

}

cout << "команда : ";

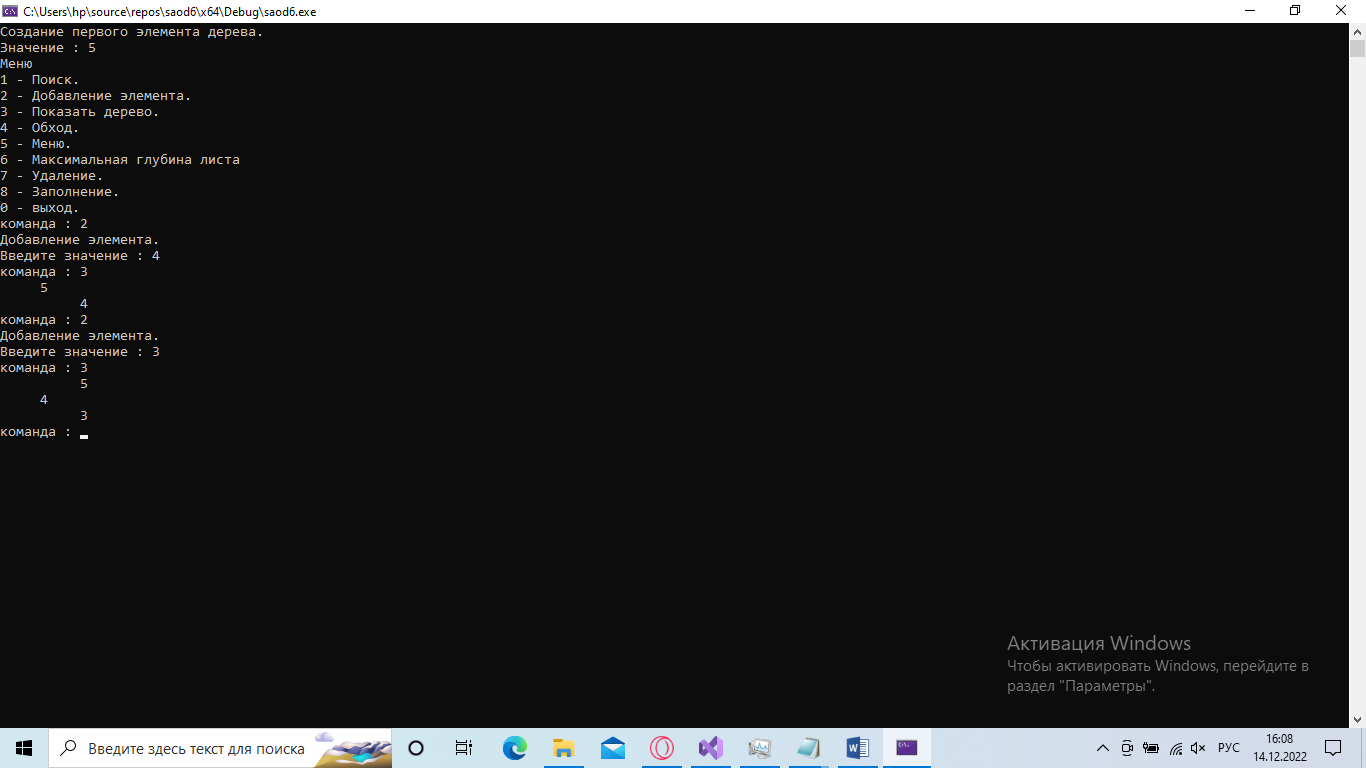
cin >> k;

}

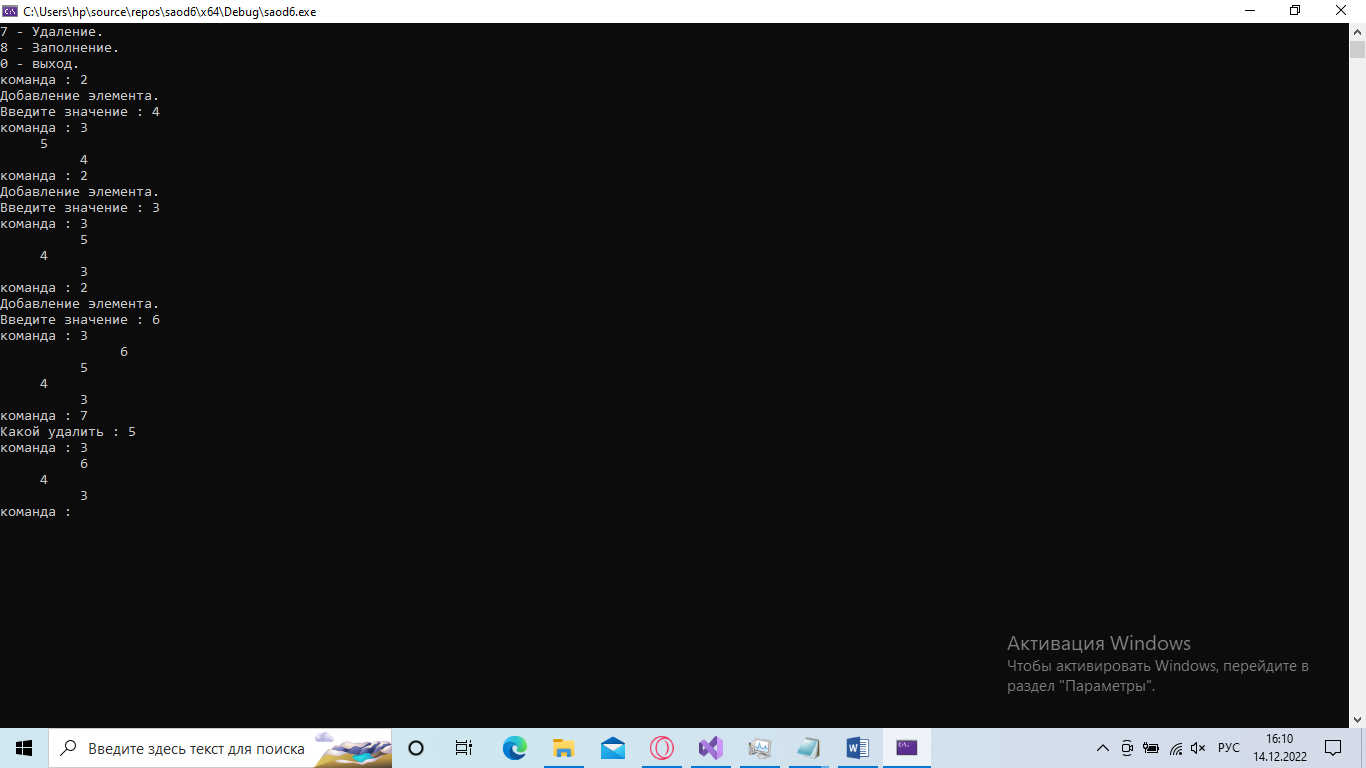
}

* 1. Работа программы

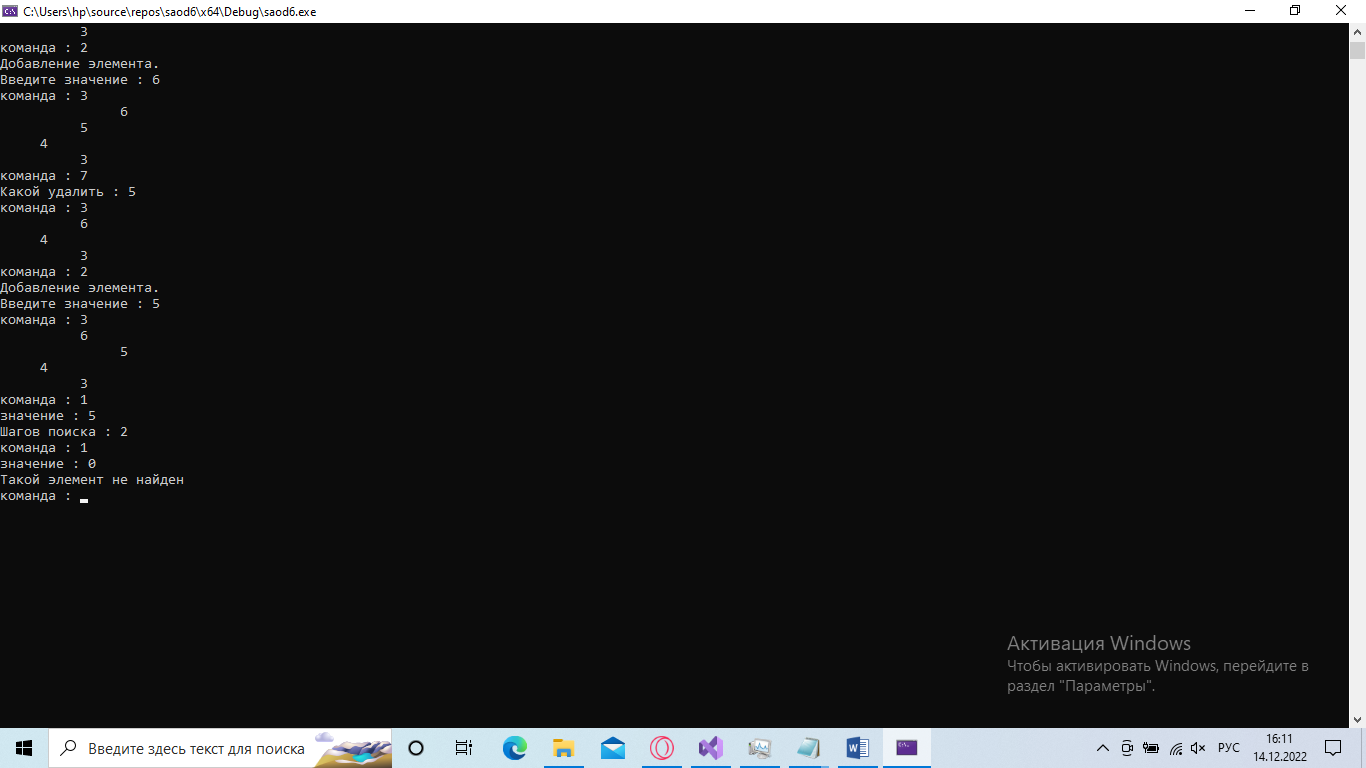
Добавление



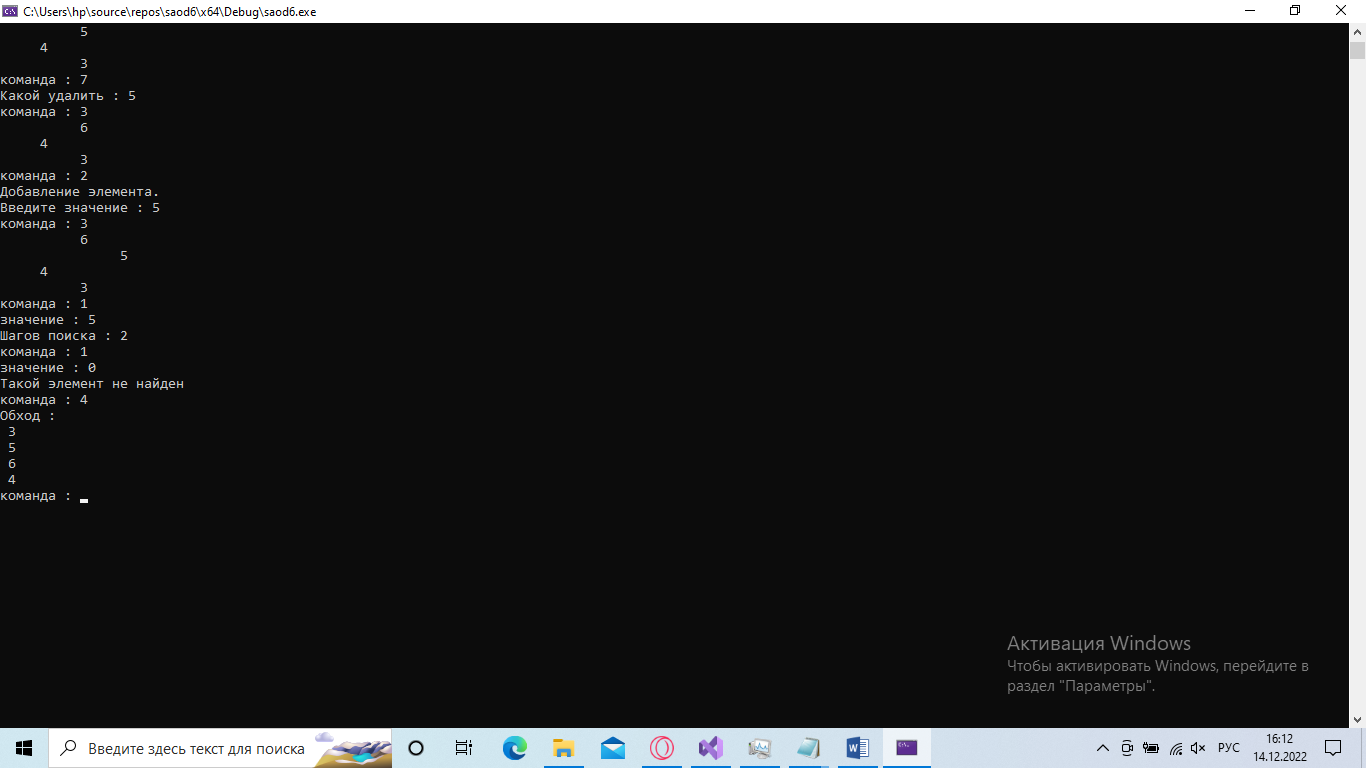
Удаление



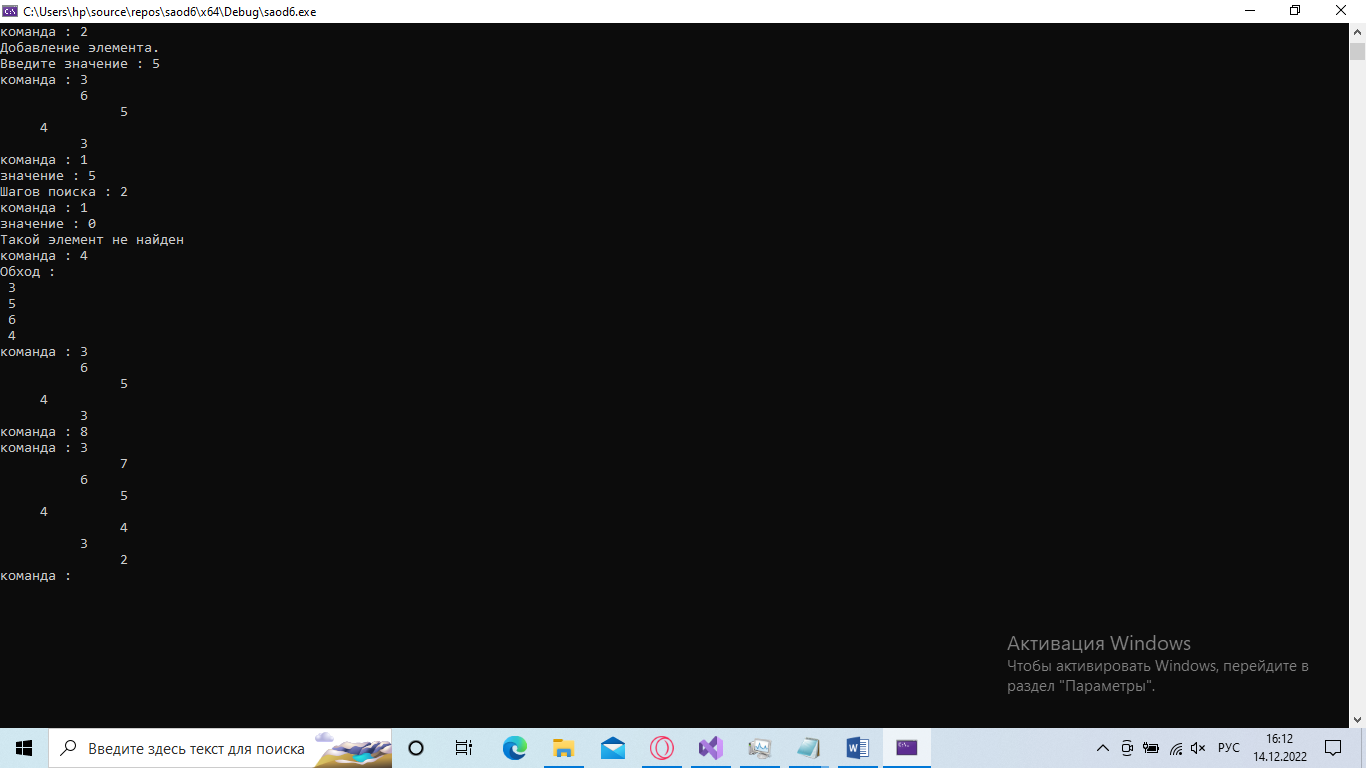
Поиск



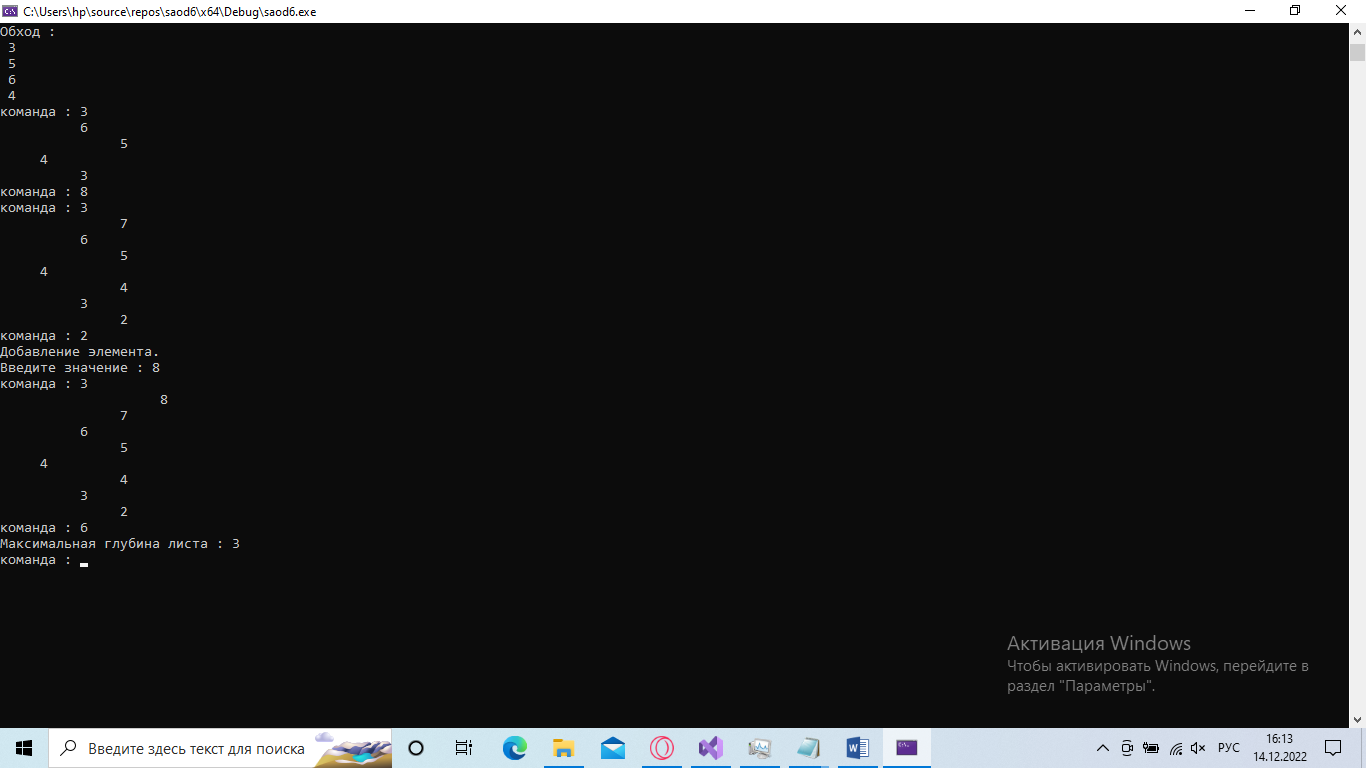
Обход



Заполнение



Максимальная глубина листа



* 1. Вывод

Я изучил деревья поиска и получил практические навыки их использования.