**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по курсовой работе**

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. | 2373 Ковальчук И. А. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2023

**Цели работы**

Изучить свойства и организации деревьев как структуры данных; получить практических навыков в работе с бинарным деревом поиска; определить преимущества и недостатки структуры данных вида дерева; проведение сравнительной характеристики скорости вставки, удаления и поиска элемента в различных структурах данных.

**Постановка задачи.**

Необходимо реализовать программу, которая выполняет следующие действия.

1.   Формирование бинарного дерева из *N* элементов:

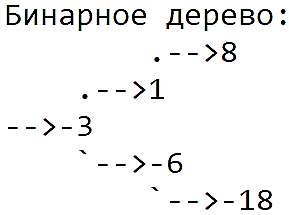
a) пользователь вводит количество элементов *N* бинарного дерева, которое автоматически заполняется случайными числами (–99 до 99);

б) пользователь вводит в консоль элементы массива, *N* определяется автоматически по количеству введенных элементов;

в) элементы считываются с файла, в котором хранится массив из чисел, *N* – количество элементов в файле.

Определение скорости формирования бинарного дерева.

2.   Вывод в консоль и файл tree бинарного дерева. Бинарное дерево должно иметь подобный вид:



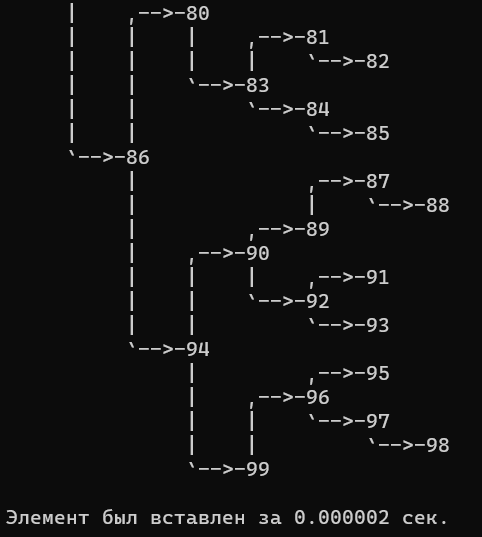
3.   Определение скорости вставки, удаления и получения элемента дерева. В отчете сравните скорость работы бинарного дерева с линейной структурой (двусвязным списком или динамическим массивом) и сделайте выводы.

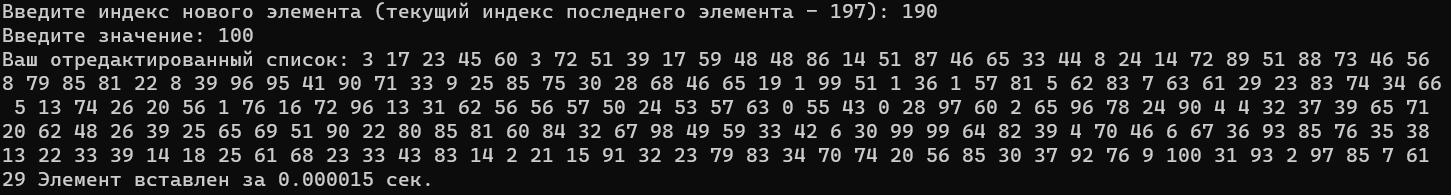
4. Прямой обход, обратный обход и обход в ширину бинарного дерева.

5.  Генерация заданий к практической работе по бинарным деревьям. Необходимо сгенерировать задания в файл output\_task в количестве вариантов, которые введет пользователь. В файл output\_key необходимо вывести короткие ответы к заданиям. В файл output\_ans необходимо вывести развернутые ответы к заданиям. Должны быть представлены задания на создание бинарного дерева поиска из массива значений, удаление и вставка элементов в бинарное дерево поиска.

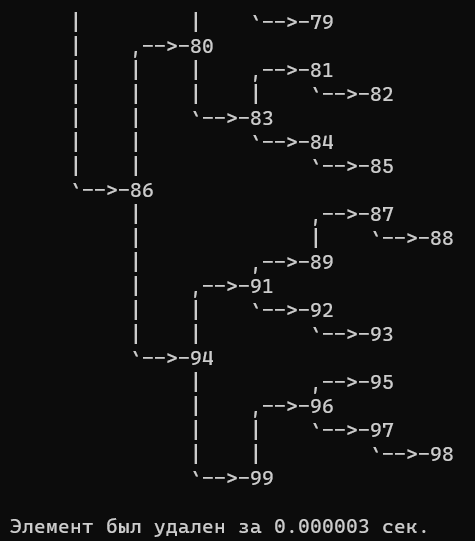
**Выполнение работы.**

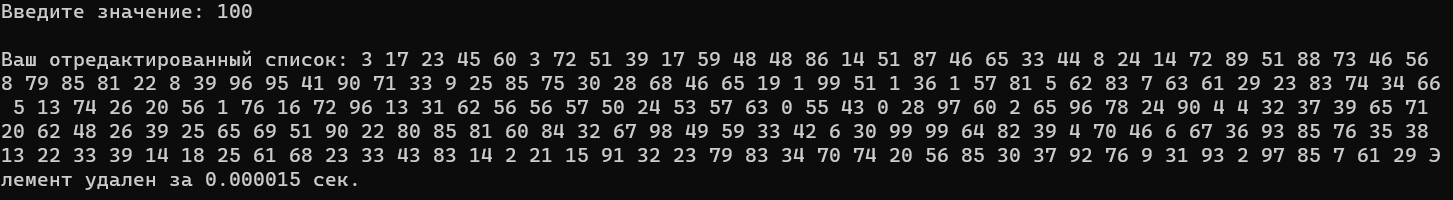
Сравнение скорости работы бинарного дерева с линейной структурой

Вставка:  
Бинарное дерево поиска:  
 

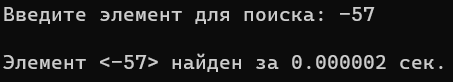
Двусвязный список:  


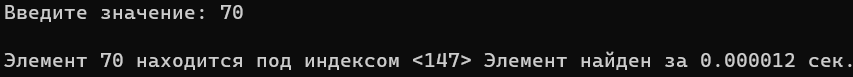
Удаление:  
Бинарное дерево поиска:



Двусвязный список:   
 Получение:

Бинарное дерево поиска:

  
Двусвязный список:

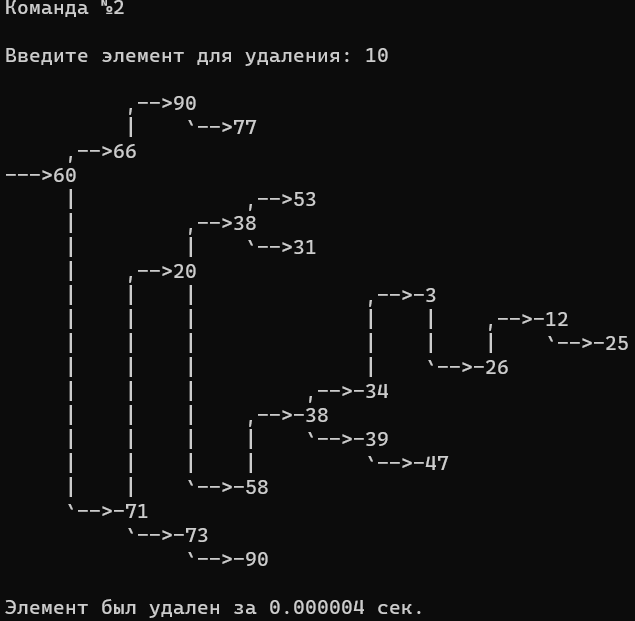
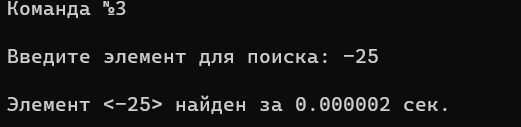
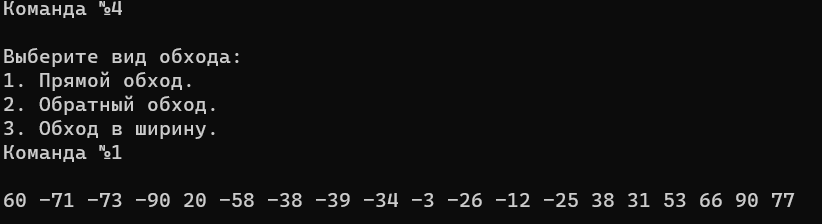


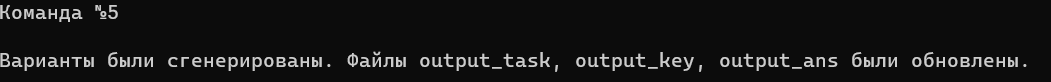
Вывод: Бинарное дерево поиска быстрее справляется со вставкой, удалением и получением элементов, чем линейная структура. С маленьким количеством элементов разница незначительная, но при работе с большим объемом данных лучше использовать бинарное дерево поиска.

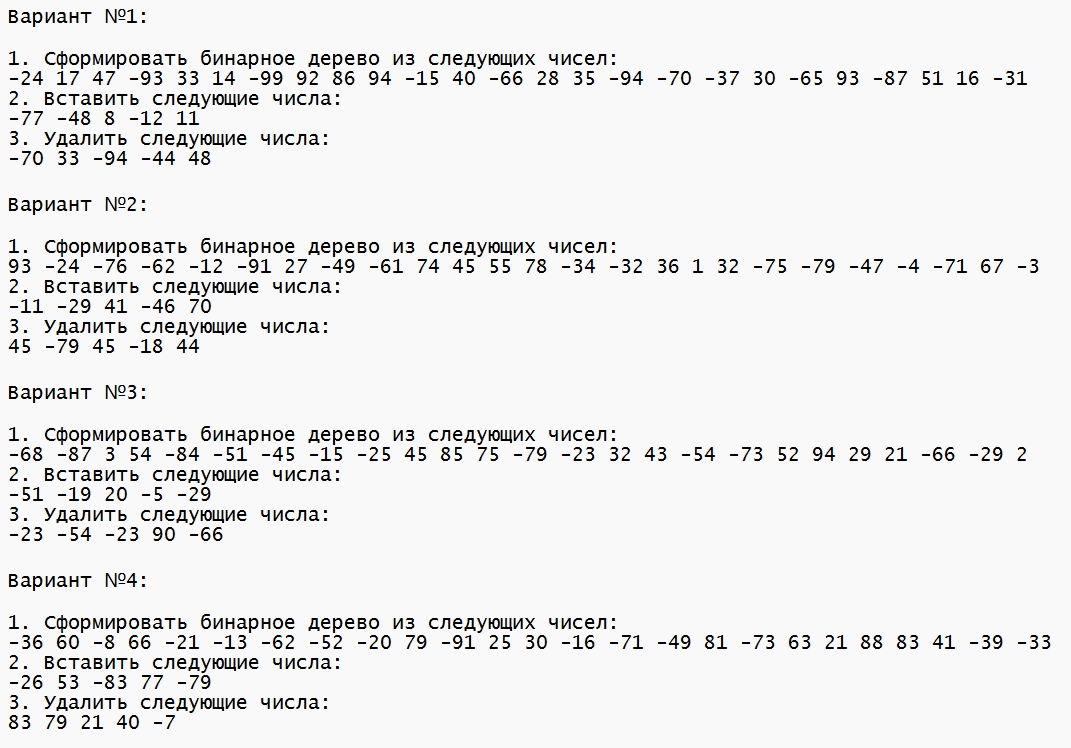
Блок скриншотов работы программы

****

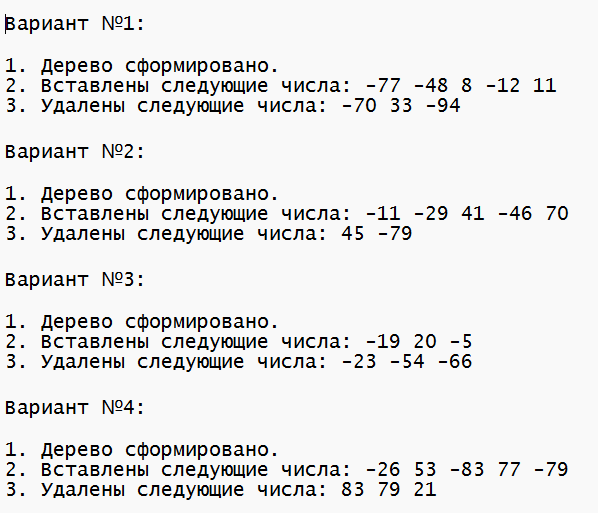
****

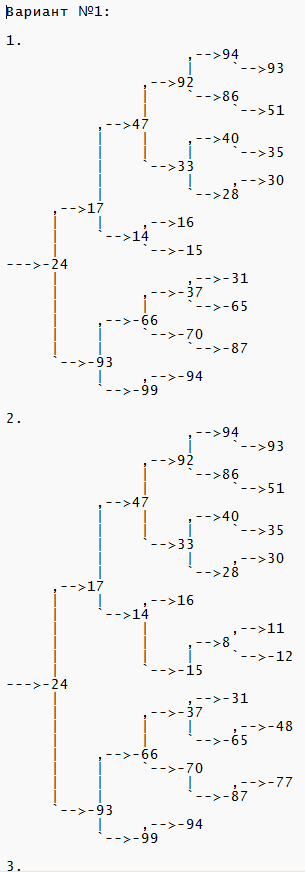
**** **** 

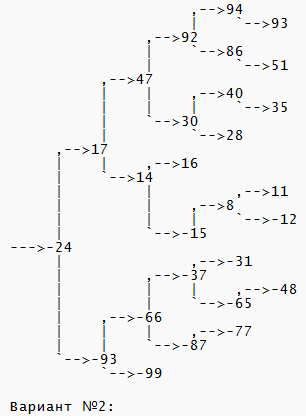
****output\_task:

****

output\_key:

  
output\_ans:

****

****

**Выводы.**

Изучены свойства и организации деревьев как структуры данных; получены практических навыков в работе с бинарным деревом поиска; определены преимущества; проведеена сравнительной характеристики скорости вставки, удаления и поиска элемента в различных структурах данных.

рабочий код

#include <iostream>

#include <fstream>

#include <iomanip>

#include <windows.h>

#include <string>

#include <chrono>

using namespace std;

// пустое дерево

struct Node {

int key;

Node\* left;

Node\* right;

};

Node\* getMaxTree(Node\* root) {

while (root->right) {

root = root->right;

}

return root;

}

int insert(Node\* root, int value) {

if (root->key == NULL) {

root->key = value;

return 1;

}

if (value < root->key) {

if (root->left == NULL) {

root->left = new Node;

root->left->key = value;

root->left->left = root->left->right = NULL;

return 1;

}

else {

insert(root->left, value);

}

}

else if (value > root->key) {

if (root->right == NULL) {

root->right = new Node;

root->right->key = value;

root->right->left = root->right->right = NULL;

return 1;

}

else {

insert(root->right, value);

}

}

else {

return 0;

}

}

int deleteElem(Node\* &root, int value) {

if (value < root->key) {

if (root->left == NULL) {

return 0;

} else if (root->left->key == value) { // found del

if (root->left->left != NULL && root->left->right != NULL) { // 2 children

int tmp = getMaxTree(root->left->left)->key;

deleteElem(root->left, tmp);

root->left->key = tmp;

return 1;

} else if (root->left->left != NULL || root->left->right != NULL) { // 1 child

if (root->left->left != NULL) { // left child of del

Node\* tmp = root->left->left;

delete root->left;

root->left = tmp;

return 1;

}

else { //right child of del

Node\* tmp = root->left->right;

delete root->left;

root->left = tmp;

return 1;

}

} else { // 0 children

delete root->left;

root->left = NULL;

return 1;

}

} else{

deleteElem(root->left, value);

}

} else if (value > root->key) {

if (root->right == NULL) {

return 0;

} else if (root->right->key == value) { // found del

if (root->right->left != NULL && root->right->right != NULL) { // 2 children

int tmp = getMaxTree(root->right->left)->key;

deleteElem(root->right, tmp);

root->right->key = tmp;

return 1;

}

else if (root->right->left != NULL || root->right->right != NULL) { // 1 child

if (root->right->left != NULL) { // left child of del

Node\* tmp = root->right->left;

delete root->right;

root->right = tmp;

return 1;

}

else { //right child of del

Node\* tmp = root->right->right;

delete root->right;

root->right = tmp;

return 1;

}

}

else { // 0 children

delete root->right;

root->right = NULL;

return 1;

}

}

else {

deleteElem(root->right, value);

}

} else { //корень дерева

if (root->left != NULL && root->right != NULL) {

int tmp = getMaxTree(root->left)->key;

deleteElem(root, tmp);

root->key = tmp;

return 1;

}else if (root->left != NULL || root->right != NULL) {

if (root->left != NULL) {

root = root->left;

} else{

root = root->right;

}

}

else {

root->key = NULL;

}

}

}

int searchElem(Node\* root, int value) {

if (value < root->key) {

if (root->left != NULL) searchElem(root->left, value);

else {

return 0;

}

}

else if (value > root->key) {

if (root->right !=NULL) searchElem(root->right, value);

else {

return 0;

}

}

else {

return 1;

}

}

void treePrint(Node\* root, string indent, int rc, int lc, short int isRight, short int lead, bool inner, short int prevlead, ofstream& Tree, bool isShown) {

if (root) {

if (lead == 1 && inner == 0 && lc != 0) {

inner = true;

}

if (lead == 0 && inner == 0 && rc != 0) {

inner = true;

}

if (rc == 1 && lc == 0 && lead == -1) lead = 1;

else if (rc == 0 && lc == 1 && lead == -1) lead = 0;

if (rc >= 1 && lc >= 1) {

indent = indent + "| ";

rc = 0;

lc = 0;

prevlead = lead;

lead = -1;

} else {

if (indent == "1") {

indent = "";

} else if (prevlead==1 && rc == 1 && lc == 0 && inner==1) {

indent = indent + "| ";

}else if(prevlead == 0 && lc == 1 && rc == 0 && inner==1) {

indent = indent + "| ";

} else indent = indent + " ";

}

treePrint(root->right, indent, rc + 1, lc, 1, lead, inner, prevlead, Tree, isShown);

//cout << root->key << " c:" << c << " lc:" << lc << " rc:" << rc << endl;

if (isRight == 1) {

if (isShown) cout << indent << ",-->" << setw(3) << left << root->key << endl;

Tree << indent << ",-->" << setw(3) << left << root->key << endl;

} else if(isRight==0){

if (isShown) cout << indent << "`-->" << setw(3) << left << root->key << endl;

Tree << indent << "`-->" << setw(3) << left << root->key << endl;

}

else {

if (root->key == NULL) {

if (isShown) cout << indent << "--->" << " " << endl;

Tree << indent << "--->" << " " << endl;

}

else {

if (isShown) cout << indent << "--->" << setw(3) << left << root->key << endl;

Tree << indent << "--->" << setw(3) << left << root->key << endl;

}

}

treePrint(root->left, indent, rc, lc+1, 0, lead, inner,prevlead, Tree, isShown);

}

}

int isUnique(int\* arr, int n) {

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

if (arr[i] == arr[n]) return 0;

}

return 1;

}

void preOrderTraversal(Node\* root) {

if (root) {

cout << root->key << " ";

preOrderTraversal(root->left);

preOrderTraversal(root->right);

}

}

void postOrderTraversal(Node\* root) {

if (root) {

postOrderTraversal(root->left);

postOrderTraversal(root->right);

cout << root->key << " ";

}

}

int height(Node\* root) {

if (root == NULL) return 0;

else {

int lh = height(root->left);

int rh = height(root->right);

if (lh > rh) return (lh + 1);

else return (rh + 1);

}

}

void levelOrderTraversal(Node\* root, int level) {

if (root == NULL) return;

if (level == 1) cout << root->key << " ";

else if (level > 1) {

levelOrderTraversal(root->left, level - 1);

levelOrderTraversal(root->right, level - 1);

}

}

int main(){

setlocale(LC\_CTYPE, ".1251");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(0));

chrono::duration<float> duration;

unsigned int choice;

cout << "Выберите способ формирования дерева:\n1. Случайно заполнить N элементов бинарного дерева.\n2. Ввести элементы через консоль.\n3. Считать элементы с файла.\nКоманда №";

cin >> choice;

int k = 0;

int\* arr = new int[200];

ifstream File;

ofstream Tree;

ofstream Keys;

ofstream Ans;

ofstream Task;

Node\* rootgen = new Node;

int ngen = 25;

int\* arrgen = new int[ngen];

int\* temparrgen = new int[ngen];

if (choice == 1) {

while (k > 199 || k < 1) {

cout << "Введите количество элементов (max 199): ";

cin >> k;

if (k > 199 || k < 1) {

cout << "Количество элементов введено неверно, попробуйте еще раз." << endl << endl;

}

}

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

arr[i] = rand() % 199 - 99;

while (!isUnique(arr, i)) {

arr[i] = rand() % 199 - 99;

}

}

cout << "Ваш список случайных элементов: ";

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

cout << arr[i] << " ";

}

cout << endl << endl;

}

else if (choice == 2) {

bool fl = true;

while (fl) {

cout << "Введите элементы через пробел, чтобы закончить ввод - нажмите Enter: (max 199, [-99, 99]): ";

int i = 0;

char ch;

while (cin >> arr[i]) {

i++;

ch = getchar();

if (ch == '\n') break;

}

k = i;

fl = false;

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

if (arr[i] > 99 || arr[i] < -99) {

cout << "В вашей последовательности существуют элементы, выходящие из диапазона [-99, 99]. Попробуйте еще раз." << endl << endl;

fl = true;

break;

}

}

if (fl) continue;

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

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

if (arr[i] == arr[j]) {

cout << "В вашей последовательности существуют одинаковые элементы. Попробуйте еще раз." << endl << endl;

fl = true;

break;

}

}

if (fl) break;

}

}

}

else if (choice == 3) {

File.open("input.txt");

if (!File.is\_open()) {

cout << "Открыть файл не удалось! \n";

return 0;

}

int temp = 999;

while (!File.eof()) {

File >> temp;

k++;

}

File.close();

if (temp == 999) {

cout << "Файл пуст.";

return 0;

}

File.open("input.txt");

if (!File.is\_open()) {

cout << "Открыть файл не удалось! \n";

return 0;

}

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

File >> arr[i];

}

File.close();

} else {

cout << "Команда введена неверно.";

return 0;

}

auto start = chrono::high\_resolution\_clock::now();

Node\* root = new Node;

root->key = arr[0];

root->left = root->right = NULL;

for (int i = 1; i < k; i++) {

insert(root, arr[i]);

}

auto end = chrono::high\_resolution\_clock::now();

duration = end - start;

Tree.open("output.txt");

treePrint(root, "1", 0, 0, -1, -1, 0, -1, Tree, 1);

Tree.close();

cout << "\nДерево сформировано за " << fixed << duration.count() << " сек." << endl;

int x, h, i, temp;

while (true) {

cout << "\nВыберите действие для работы с деревом:\n1. Вставить новый элемент\n2. Удалить существующий элемент\n3. Получить элемент\n4. Выполнить обход по дереву.\n5. Сгенерировать варианты заданий.\nКоманда №";

cin >> choice;

cout << endl;

switch (choice){

case 1:

while (true) {

cout << "Введите элемент для вставки: ";

cin >> x;

cout << endl;

if (x <= 99 && x >= -99) {

auto start = chrono::high\_resolution\_clock::now();

temp = insert(root, x);

auto end = chrono::high\_resolution\_clock::now();

duration = end - start;

break;

}

else {

cout << "Элемент не принадлежит диапазону [-99, 99]. Попробуйте еще раз." << endl << endl;

}

}

Tree.open("output.txt");

treePrint(root, "1", 0, 0, -1, -1, 0, -1, Tree, 1);

Tree.close();

if (temp) cout << "\nЭлемент был вставлен за " << fixed << duration.count() << " сек." << endl;

else cout << "Дерево не изменилось. Такой элемент уже существует." << endl << endl;

break;

case 2:

while (true) {

cout << "Введите элемент для удаления: ";

cin >> x;

cout << endl;

if (x <= 99 && x >= -99) {

auto start = chrono::high\_resolution\_clock::now();

temp = deleteElem(root, x);

auto end = chrono::high\_resolution\_clock::now();

duration = end - start;

break;

}

else {

cout << "Элемент не принадлежит диапазону [-99, 99]. Попробуйте еще раз." << endl << endl;

}

}

Tree.open("output.txt");

treePrint(root, "1", 0, 0, -1, -1, 0, -1, Tree, 1);

Tree.close();

if (temp) {

cout << "\nЭлемент был удален за " << fixed << duration.count() << " сек." << endl;

}

else {

cout << "\nТакого элемента не существует." << endl;

}

break;

case 3:

while (true) {

cout << "Введите элемент для поиска: ";

cin >> x;

cout << endl;

if (x <= 99 && x >= -99) {

auto start = chrono::high\_resolution\_clock::now();

temp = searchElem(root, x);

auto end = chrono::high\_resolution\_clock::now();

duration = end - start;

break;

}

else {

cout << "Элемент не принадлежит диапазону [-99, 99]. Попробуйте еще раз." << endl << endl;

}

}

if (temp) cout << "Элемент <" << x << "> найден за " << fixed << duration.count() << " сек." << endl;

else cout << "Такого элемента не существует" << endl;

break;

case 4:

cout << "Выберите вид обхода:\n1. Прямой обход.\n2. Обратный обход.\n3. Обход в ширину.\nКоманда №";

cin >> choice;

cout << endl;

switch (choice){

case 1:

preOrderTraversal(root);

cout << endl;

break;

case 2:

postOrderTraversal(root);

cout << endl;

break;

case 3:

h = height(root);

for (i = 1; i <= h; i++) levelOrderTraversal(root, i);

cout << endl;

break;

default:

break;

}

break;

case 5:

cout << "Варианты были сгенерированы. Файлы output\_task, output\_key, output\_ans были обновлены." << endl;

Task.open("output\_task.txt", ofstream::out | ofstream::trunc);

Task.close();

Keys.open("output\_key.txt", ofstream::out | ofstream::trunc);

Keys.close();

Ans.open("output\_ans.txt", ofstream::out | ofstream::trunc);

Ans.close();

for (int k = 1; k < 11; k++) {

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

arrgen[i] = rand() % 199 - 99;

while (!isUnique(arrgen, i)) {

arrgen[i] = rand() % 199 - 99;

}

}

Task.open("output\_task.txt", ios::app);

Task << "Вариант №" << k << ":\n\n1. Сформировать бинарное дерево из следующих чисел:\n";

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

Task << arrgen[i] << " ";

}

Task << endl;

Task.close();

rootgen->key = arrgen[0];

rootgen->left = rootgen->right = NULL;

for (int i = 1; i < ngen; i++) {

insert(rootgen, arrgen[i]);

}

Keys.open("output\_key.txt", ios::app);

Keys << "Вариант №" << k << ":\n\n1. Дерево сформировано.\n";

Keys.close();

Ans.open("output\_ans.txt", ios::app);

Ans << "Вариант №" << k << ":\n\n1.\n";

treePrint(rootgen, "1", 0, 0, -1, -1, 0, -1, Ans, 0);

Ans.close();

//2

Task.open("output\_task.txt", ios::app);

Task << "2. Вставить следующие числа:\n";

for (int i = 0; i < 5; i++) {

temparrgen[i] = rand() % 199 - 99;

Task << temparrgen[i] << " ";

}

Task.close();

Keys.open("output\_key.txt", ios::app);

Keys << "2. Вставлены следующие числа: ";

for (int i = 0; i < 5; i++) {

if (insert(rootgen, temparrgen[i])) Keys << temparrgen[i] << " ";

}

Keys.close();

Ans.open("output\_ans.txt", ios::app);

Ans << "\n2.\n";

treePrint(rootgen, "1", 0, 0, -1, -1, 0, -1, Ans, 0);

Ans.close();

//3

Task.open("output\_task.txt", ios::app);

Task << "\n3. Удалить следующие числа:\n";

for (int i = 0; i < 3; i++) {

temparrgen[i] = arrgen[(rand() % 25)];

Task << temparrgen[i] << " ";

}

for (int i = 3; i < 5; i++) {

temparrgen[i] = rand() % 199 - 99;

Task << temparrgen[i] << " ";

}

Task << endl << endl;

Task.close();

Keys.open("output\_key.txt", ios::app);

Keys << "\n3. Удалены следующие числа: ";

for (int i = 0; i < 5; i++) {

if (deleteElem(rootgen, temparrgen[i])) Keys << temparrgen[i] << " ";

}

Keys << endl << endl;

Keys.close();

Ans.open("output\_ans.txt", ios::app);

Ans << "\n3.\n";

treePrint(rootgen, "1", 0, 0, -1, -1, 0, -1, Ans, 0);

Ans << endl;

Ans.close();

}

default:

break;

}

}

}