Министерство образования и науки Российской Федерации

ФГБОУ ВО «Пензенский государственный университет»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Пенза 2025

**Отчет**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнил: студент группы 24ВВВ1

Егорова К.А.

Винникова Е.С

Марушкин Д.И.

Проверил: к.т.н., доцент

Юрова О.В.

Деев М.В.

**Лабораторные задания:**

1. Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.

2. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

3. Изменить функцию добавления элементов для исключения добавления одинаковых символов.

4. Оценить сложность процедуры поиска по значению в бинарном дереве.

**Пояснительный текст к программе:**

**1. Назначение программы:**

Цель программы — изучение структуры данных «бинарное дерево поиска», освоение операций добавления, поиска, подсчёта повторов и предотвращения вставки дубликатов.

Программа позволяет создавать бинарное дерево из любых типов данных, используя шаблонный класс, искать заданное число в дереве, запрещать добавление одинаковых данных, а так же выводить данные в консоль.

**2. Структура программы:**

BinTree — шаблонный класс бинарного дерева с методами insert(), search(), countEntry(), print().

Вложенная структура Node хранит значение узла и указатели на левый и правый потомки.

main() реализует взаимодействие с пользователем: ввод чисел, построение дерева, поиск, подсчёт вхождений и вывод дерева.

**3. Используемые средства:**

Язык C++ (шаблоны, динамическое выделение памяти).

Рекурсивные функции для работы с деревом.

Стандартные библиотеки <iostream>, <locale>, <string>, <cctype>.

**4. Результат работы программы:**

Измерить время поиска значения в бинарном дереве поиска в двух случаях:

1. Сбалансированное дерево - числа вставлены в случайном порядке.

2. Вырожденное дерево - числа вставлены в возрастающем порядке.

Для каждого размера дерева количество элементов n=100, 200, 400, 800, 1600 создаём отдельное дерево:

Вырожденное — вставляем числа от 1 до n по порядку.

Случайное — вставляем те же числа в случайном порядке.

Для каждого дерева ищем последний элемент (максимальное значение) и фиксируем время поиска.

Таблица 1.

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | Время вырожденного дерева (мкс) | Время случайного дерева (мкс) |
| 100 | 8 | 1 |
| 200 | 13 | 0 |
| 400 | 19 | 1 |
| 800 | 37 | 1 |
| 1600 | 131 | 1 |

Диаграмма 1.

Поиск в сбалансированном BST выполняется за O(log n), в худшем случае (вырожденное дерево-список) - O(n).

**Трассировка программы:**

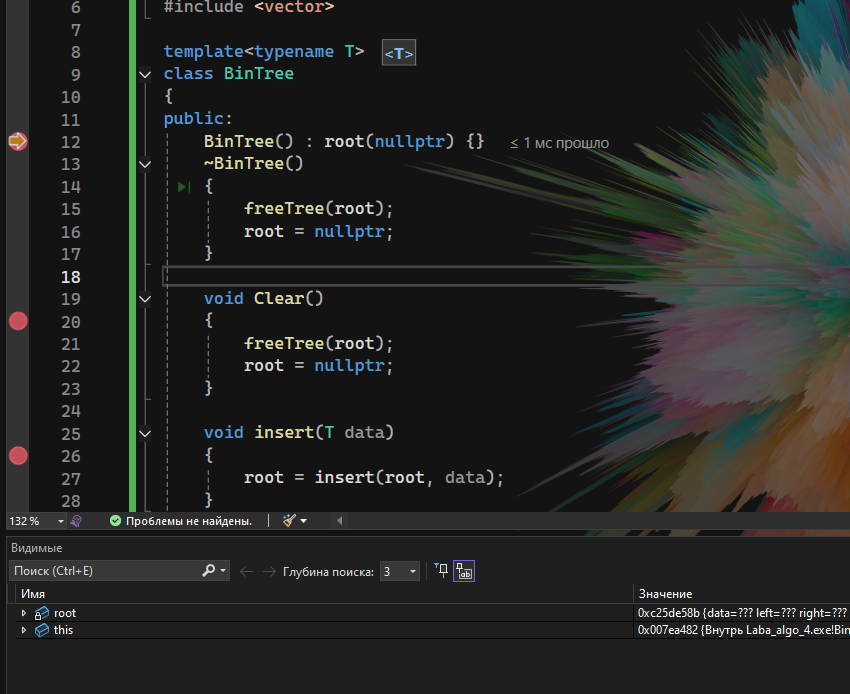


Рисунок 1. Создание объекта, используя конструктор по умолчанию.

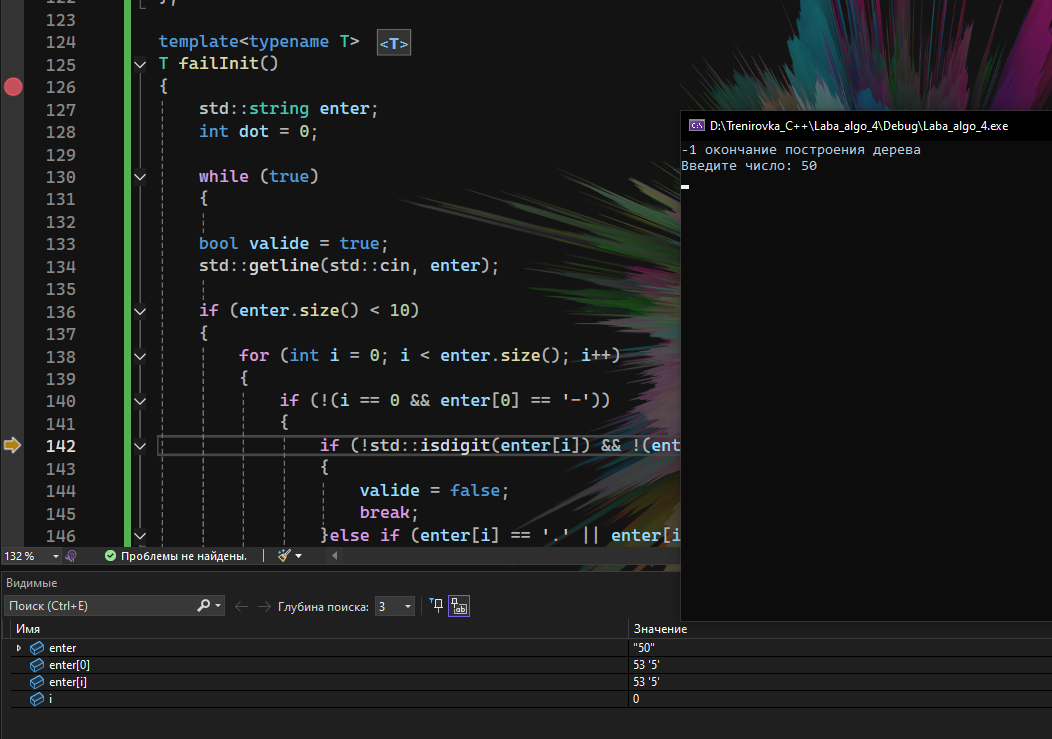


Рисунок 2 - Переход в функцию проверки вводимого числа.

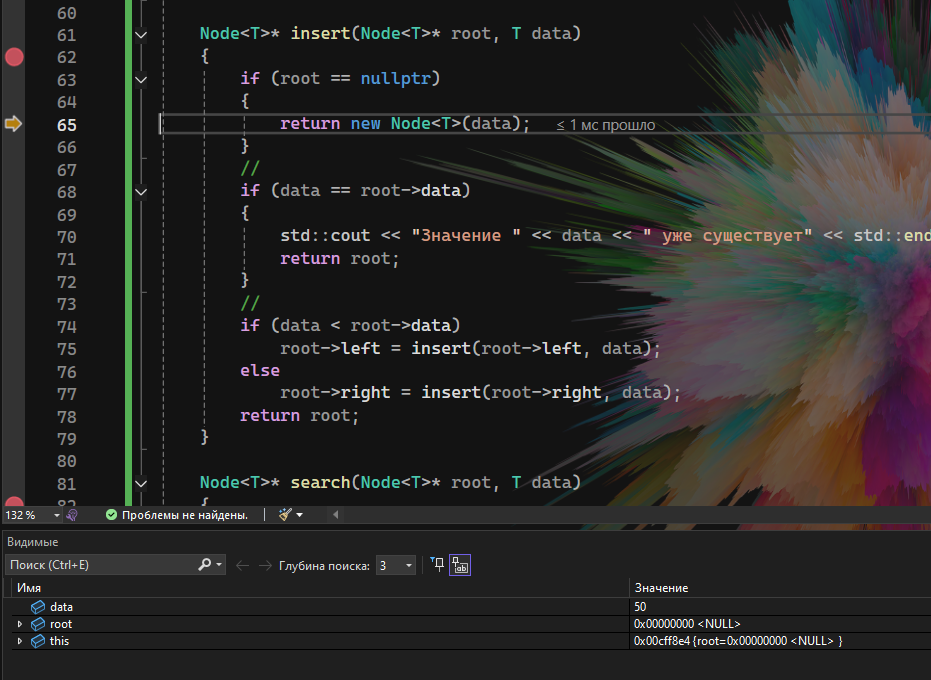


Рисунок 3 - Переход в метод создания объекта со входными данными data.

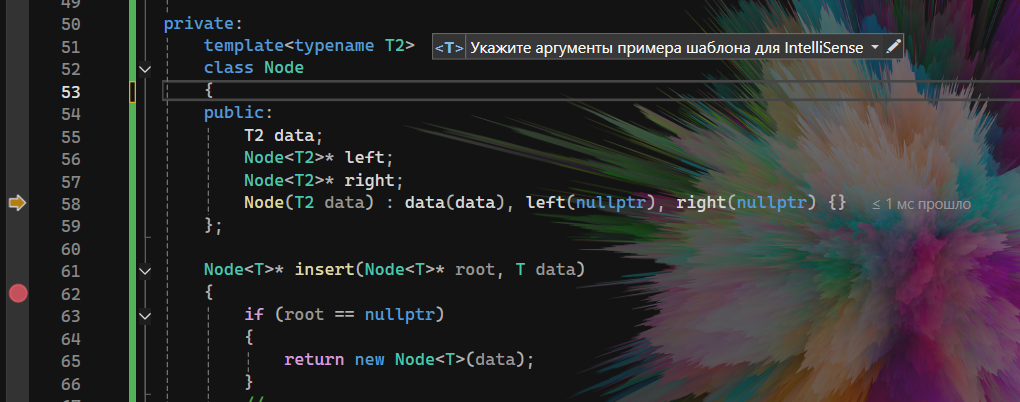


Рисунок 4 - Создание и инициализация объекта Node используя конструктор.

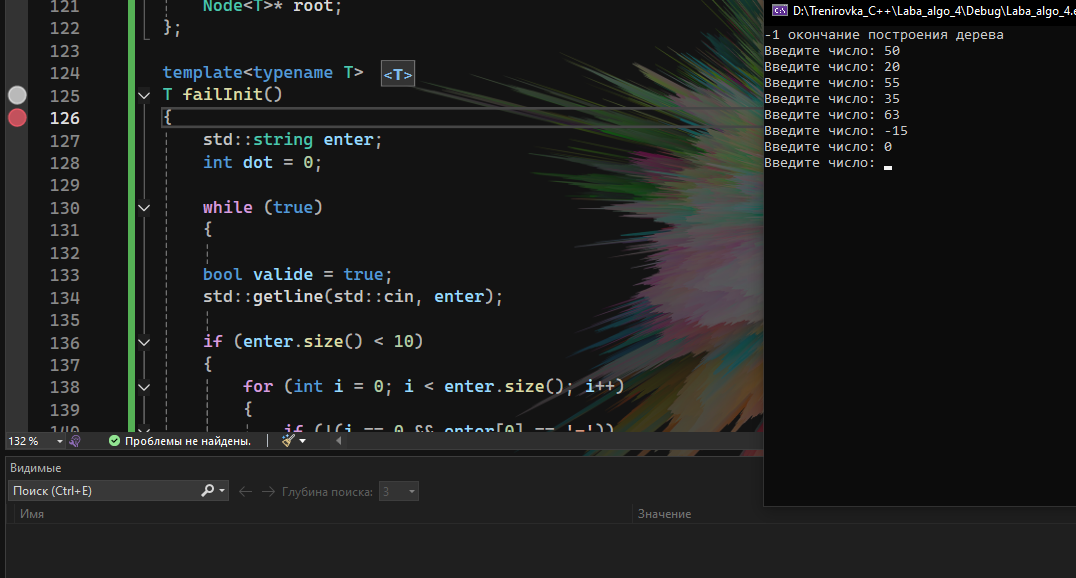


Рисунок 5 - Добавление других значений.

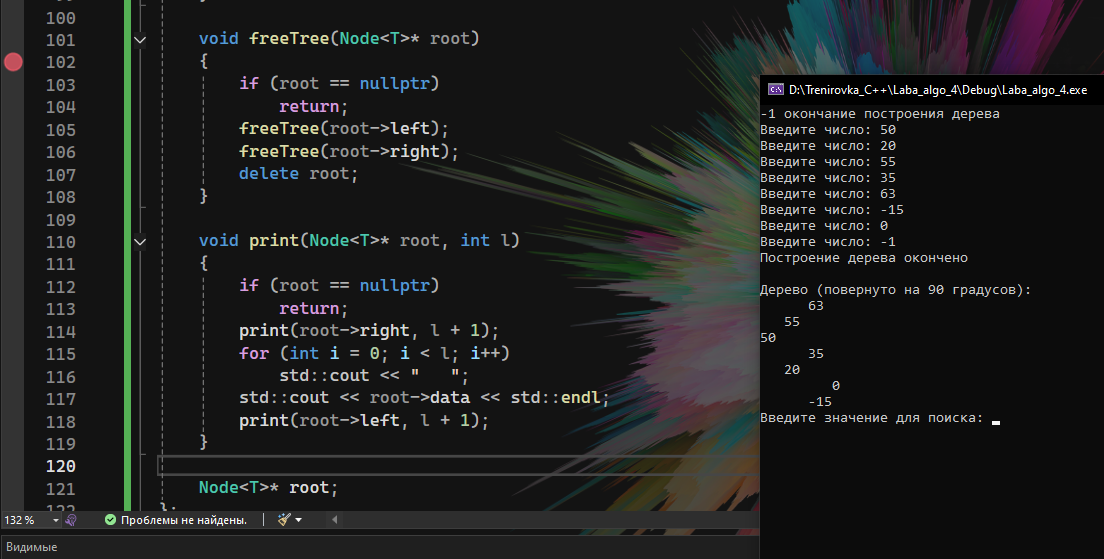


Рисунок 6 - Вывод дерева в консоль.

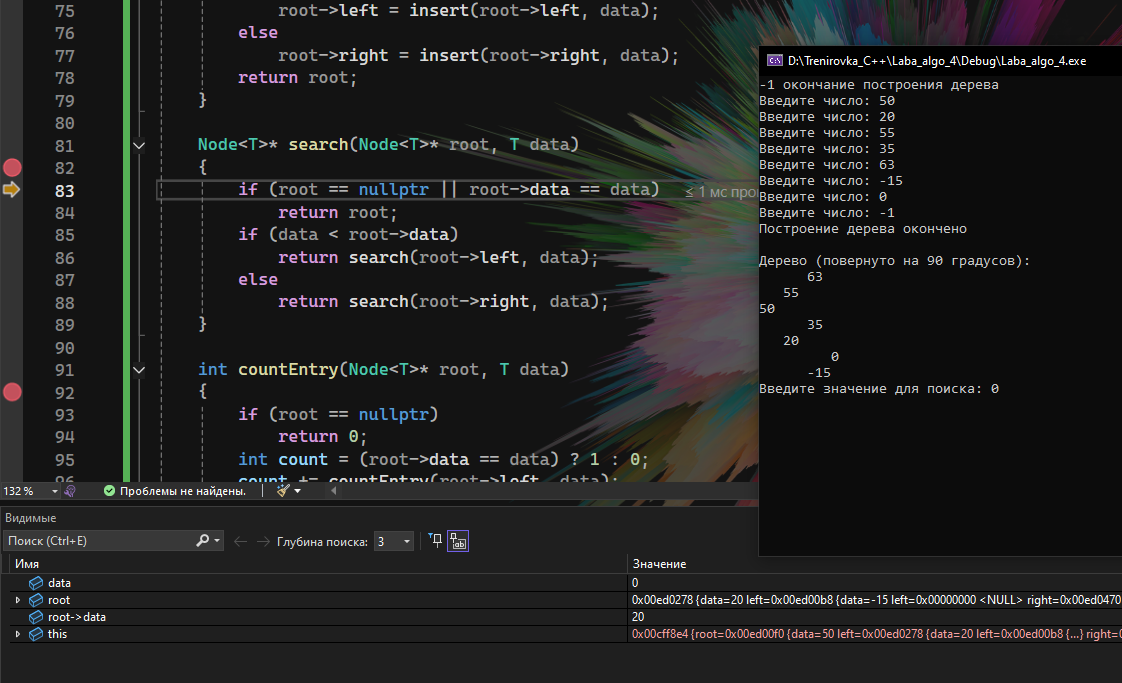


Рисунок 7 - Поиск значения в дереве.

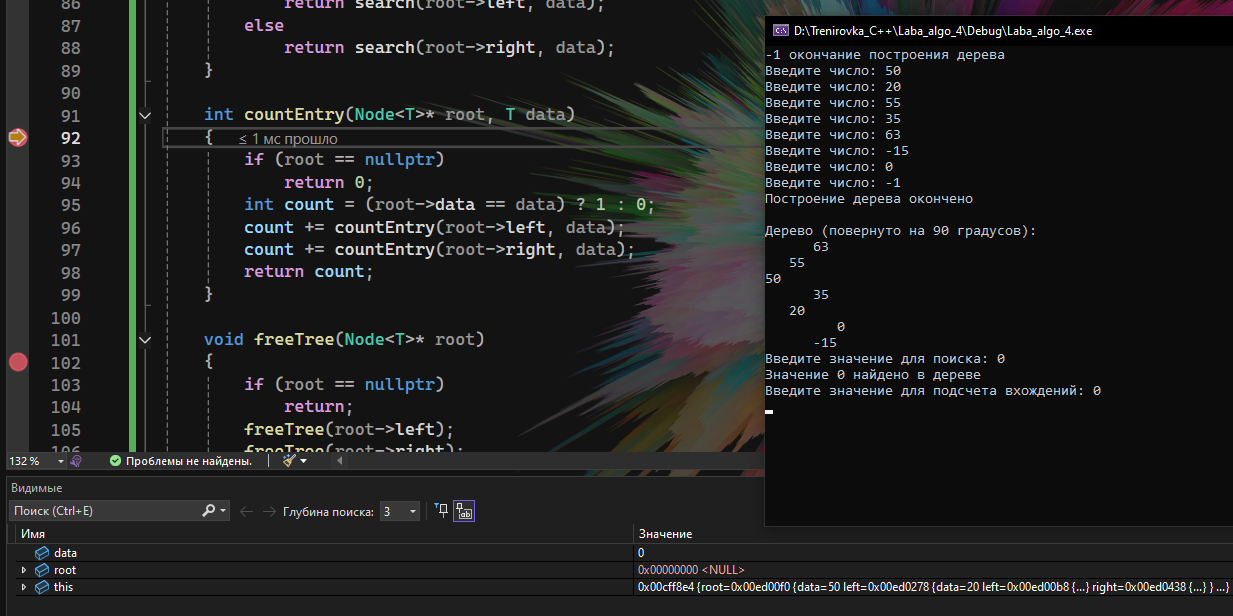


Рисунок 8 - Подсчет числа вхождений элемента.

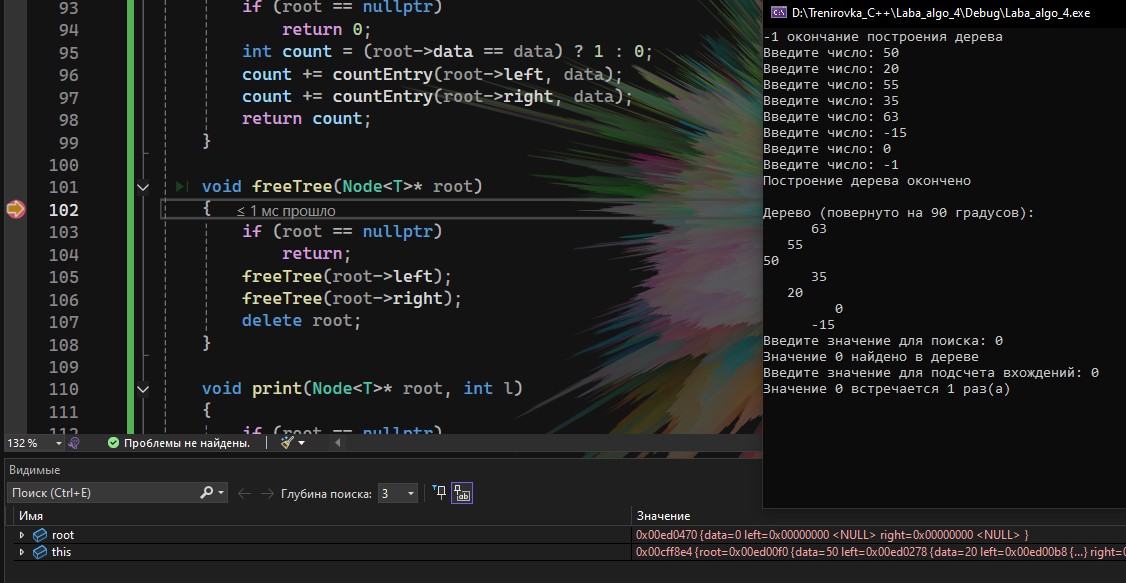


Рисунок 9 – Вызов деструктора, для освобождения памяти.

**Результат работы программы:**

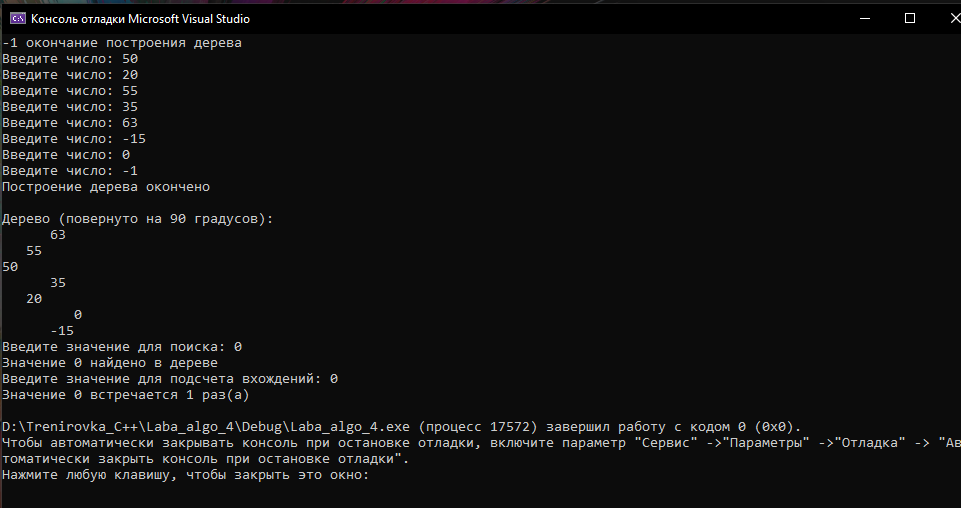


Рисунок 10 – Результат работы программы.

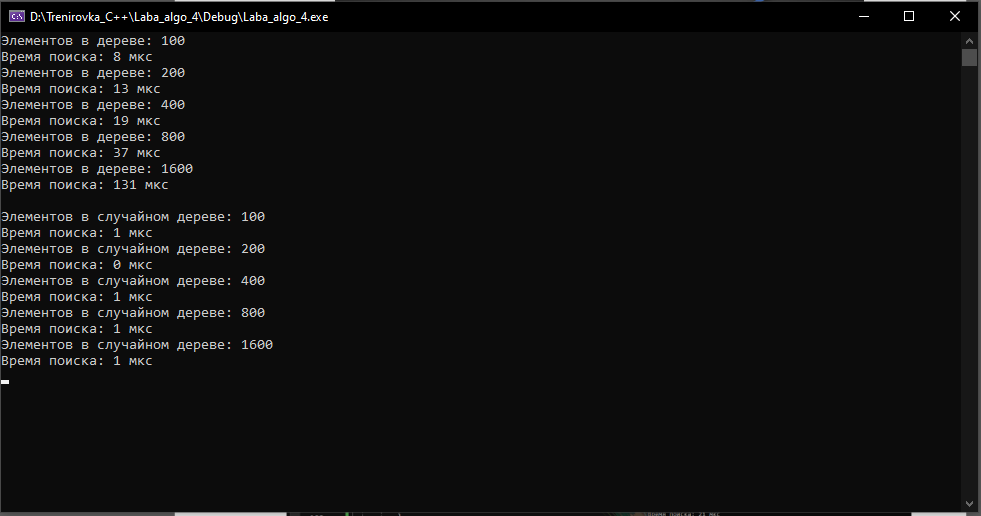


Рисунок 11 – Вывод затраченного времени на поиск элементов.

**Листинг программы:**

#include <iostream>

#include <cctype>

#include <string>

#include <chrono>

#include <random>

#include <vector>

template<typename T>

class BinTree

{

public:

BinTree() : root(nullptr) {}

~BinTree()

{

freeTree(root);

root = nullptr;

}

void Clear()

{

freeTree(root);

root = nullptr;

}

void insert(T data)

{

root = insert(root, data);

}

void print()

{

if (root == nullptr)

{

std::cout << "Дерево пусто" << std::endl;

return;

}

print(root, 0);

}

bool search(T data)

{

return search(root, data) != nullptr;

}

int countEntry(T data)

{

return countEntry(root, data);

}

private:

template<typename T2>

class Node

{

public:

T2 data;

Node<T2>\* left;

Node<T2>\* right;

Node(T2 data) : data(data), left(nullptr), right(nullptr) {}

};

Node<T>\* insert(Node<T>\* root, T data)

{

if (root == nullptr)

{

return new Node<T>(data);

}

//

if (data == root->data)

{

std::cout << "Значение " << data << " уже существует" << std::endl;

return root;

}

//

if (data < root->data)

root->left = insert(root->left, data);

else

root->right = insert(root->right, data);

return root;

}

Node<T>\* search(Node<T>\* root, T data)

{

if (root == nullptr || root->data == data)

return root;

if (data < root->data)

return search(root->left, data);

else

return search(root->right, data);

}

int countEntry(Node<T>\* root, T data)

{

if (root == nullptr)

return 0;

int count = (root->data == data) ? 1 : 0;

count += countEntry(root->left, data);

count += countEntry(root->right, data);

return count;

}

void freeTree(Node<T>\* root)

{

if (root == nullptr)

return;

freeTree(root->left);

freeTree(root->right);

delete root;

}

void print(Node<T>\* root, int l)

{

if (root == nullptr)

return;

print(root->right, l + 1);

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

std::cout << " ";

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

print(root->left, l + 1);

}

Node<T>\* root;

};

template<typename T>

T failInit()

{

std::string enter;

int dot = 0;

while (true)

{

bool valide = true;

std::getline(std::cin, enter);

if (enter.size() < 10)

{

for (int i = 0; i < enter.size(); i++)

{

if (!(i == 0 && enter[0] == '-'))

{

if (!std::isdigit(enter[i]) && !(enter[i] == '.' || enter[i] == ','))

{

valide = false;

break;

}else if (enter[i] == '.' || enter[i] == ',')

{

enter[i] = ',';

++dot;

}

else if (dot > 1)

{

valide = false;

break;

}

}

}

}

else

{

valide = false; // слишком длинная строка

}

if(valide) break;

std::cout << "Некорректный ввод" << std::endl;

std::cin.clear();

}

return static\_cast<T>(std::stod(enter));

}

void timeBt(BinTree<int>& other, int data)

{

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

other.search(data);

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

auto result = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "Время поиска: " << result.count() << " мкс" << std::endl;

}

void testBt()

{

BinTree<int> bt;

std::random\_device rd;

std::mt19937 gen(rd());

int value[5] = { 100, 200, 400, 800, 1600 };

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

{

for (int n = 0; n < value[i]; n++)

{

bt.insert(n);

}

std::cout << "Элементов в дереве: " << value[i] << std::endl;

timeBt(bt, value[i] - 1);

bt.Clear();

}

std::cout << '\n';

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

{

std::vector<int> rndValue;

std::uniform\_int\_distribution<int> dist(1, value[i]);

for (int n = 0; n < value[i]; n++)

{

rndValue.push\_back(n);

}

std::shuffle(rndValue.begin(), rndValue.end(), gen);

for (int n = 0; n < value[i]; n++)

{

bt.insert(rndValue[n]);

}

std::cout << "Элементов в случайном дереве: " << value[i] << std::endl;

timeBt(bt, value[i] - 1);

bt.Clear();

}

}

int main()

{

using Type = int;

setlocale(LC\_ALL, "RUSSIAN");

//testBt();

BinTree<Type> bt;

Type data, search\_value;

bool start = true;

std::cout << "-1 окончание построения дерева" << std::endl;

while (start)

{

std::cout << "Введите число: ";

//std::getline(std::cin, data);

data = failInit<Type>();

if (data == -1)

{

std::cout << "Построение дерева окончено\n" << std::endl;

start = false;

}

else

{

bt.insert(data);

}

}

std::cout << "Дерево (повернуто на 90 градусов): " << std::endl;

bt.print();

std::cout << "Введите значение для поиска: ";

//std::getline(std::cin, search\_value);

std::cin >> search\_value;

if (bt.search(search\_value))

std::cout << "Значение " << search\_value << " найдено в дереве" << std::endl;

else

std::cout << "Значение " << search\_value << " не найдено в дереве" << std::endl;

std::cout << "Введите значение для подсчета вхождений: ";

std::cin >> search\_value;

int count = bt.countEntry(search\_value);

std::cout << "Значение " << search\_value << " встречается " << count << " раз(а)" << std::endl;

return 0;

}