МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯРОССИЙСКОЙ ФЕДЕРАЦИИПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №4

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Бинарное дерево поиска"

Выполнили:

Студенты группы 24ВВВ3:

Плотников И.А.

Виноградов Б.С.

Приняли:

Деев М.В.

Юрова О. В.

Пенза 2025

Цель

Изучение бинарного дерева.

Лабораторное задание

Задание

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.

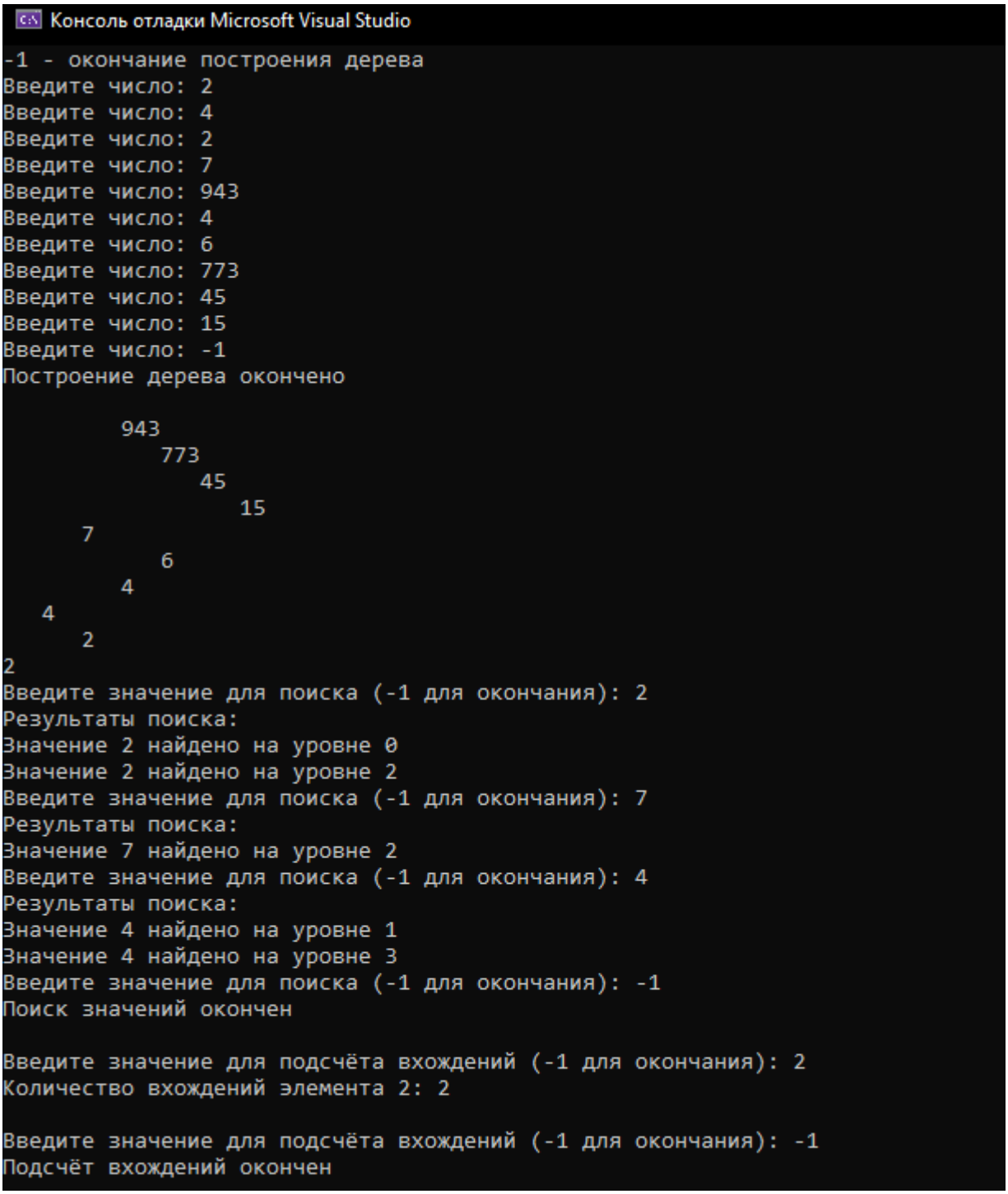
4* Оценить сложность процедуры поиска по значению в бинарном дереве.

Пояснительный текст к программам

Эта программа позволяет пользователю создать бинарное дерево поиска, вводя целые числа по одному. В процессе построения, если вводимое число равно или больше текущего узла, оно добавляется в правое поддерево, что позволяет учитывать дубликаты как “>=”. Если число меньше — оно добавляется в левое поддерево. После завершения построения дерева пользователь может просмотреть его структуру в виде горизонтального дерева, где корень — в центре, а ветви — слева и справа. Далее пользователь может искать конкретное значение: программа выводит все уровни, на которых встречается искомый элемент, учитывая дубликаты, и сообщает уровень (начиная с 0 в корне). Также есть возможность подсчета общего количества вхождений выбранного элемента в дерево. В конце программа освобождает память, удаляя все узлы дерева, чтобы избежать утечек.

Результаты программ

1 Рис. - Результат работы **lab4.cpp**



1. Вычисление порядка сложности программы (О-символика)

Процедура поиска по значению в бинарном дереве, реализованная в нашем коде, выполняется рекурсивно и выглядит следующим образом:

- CopyRunvoid searchLevel(Node* root, int value, int level = 0) {
- if (root == nullptr) return;
-
- if (root->data == value)
- std::cout << "Значение " << value << " найдено на уровне " << level << std::endl;
-
- if (value < root->data)
- searchLevel(root->left, value, level + 1);
- else
- searchLevel(root->right, value, level + 1);
- }

Анализ сложности:

- В худшем случае, если дерево является вырожденным (например, все узлы располагаются в одной ветке, образуя цепочку), поиск будет осуществляться последовательно по всей цепочке. В этом случае количество проверяемых узлов равно количеству узлов в дереве, то есть O(n), где n — количество узлов.
- В лучшем случае, если дерево идеально сбалансировано (например, дерево с минимальной высотой), глубина дерева составляет примерно $\log_2 n$. Тогда поиск в худшем случае требует проверки на уровне, равном O(log n).

Вывод: В ходе выполнения лабораторной работы была разработана программа для выполнения заданий Лабораторной работы №4 – построение бинарного дерева поиска. В процессе выполнения работы была произведена оценка сложности программы.

Приложение А

Листинг

Файл lab4.cpp

```
#include <iostream>
#include <limits>

struct Node {
    int data;
    Node* left;
    Node* right;
};

int isInteger(const std::string& message);
void clearScreen();
Node* createTree(Node* root, int data);
void printTree(Node* node, int level = 0);
void searchLevel(Node* root, int value, int level = 0);
int countOccurrences(Node* root, int value);
void deleteTree(Node* root);

int main() {
    clearScreen();
    Node* root = nullptr;
    int value;
    std::cout << "-1 — окончание построения дерева\n";

    while (true) {
        value = isInteger("Введите число: ");
        if (value == -1) {
            std::cout << "Построение дерева окончено\n\n";
            break;
        }
        root = createTree(root, value);
    }

    printTree(root);
```

```
        while (true) {
            value = isInteger("Введите значение для поиска (-1 для
окончания): ");
            if (value == -1) {
                std::cout << "Поиск значений окончен\n";
                break;
            }
            std::cout << "Результаты поиска:\n";
            searchLevel(root, value);
        }

        while (true) {
            value = isInteger("\nВведите значение для подсчёта вхождений (-1
для окончания): ");
            if (value == -1) {
                std::cout << "Подсчёт вхождений окончен\n";
                break;
            }
            std::cout << "Количество вхождений элемента " << value << ": "
                    << countOccurrences(root, value) << std::endl;
        }

        deleteTree(root);
        return 0;
    }


    int isInteger(const std::string& message) {
        int value;
        while (true) {
            std::cout << message;
            if (!(std::cin >> value)) {
                std::cout << "Ошибка: введено не число.\n";
                std::cin.clear();
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                continue;
            }
            if (std::cin.peek() != '\n') {
                std::cout << "Ошибка: введено не целое число.\n";
```

```
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            continue;
        }
        return value;
    }
}

void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

Node* createTree(Node* root, int data) {
    if (root == nullptr) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->left = nullptr;
        newNode->right = nullptr;
        return newNode;
    }


    if (data < root->data) {
        root->left = createTree(root->left, data);
    } else {
        root->right = createTree(root->right, data);
    }

    return root;
}

void printTree(Node* node, int level) {
    if (node == nullptr) return;
    printTree(node->right, level + 1);
    for (int i = 0; i < level; i++) std::cout << "    ";
```

```
        std::cout << node->data << std::endl;
        printTree(node->left, level + 1);
    }

    void searchLevel(Node* root, int value, int level) {
        if (root == nullptr) return;

        if (root->data == value)
            std::cout << "Значение " << value << " найдено на уровне " << level
<< std::endl;

        if (value < root->data)
            searchLevel(root->left, value, level + 1);
        else
            searchLevel(root->right, value, level + 1);
    }

    int countOccurrences(Node* root, int value) {
        if (root == nullptr) return 0;
        int count = 0;
        if (root->data == value) count++;
        count += countOccurrences(root->left, value);
        count += countOccurrences(root->right, value);
        return count;
    }

    void deleteTree(Node* root) {
        if (root == nullptr) return;
        deleteTree(root->left);
        deleteTree(root->right);
        delete root;
    }
```