

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ПЕНЗЕНСКИЙ 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
24ВВВЗ:

Пяткин Р. С.

Гусаров Е. Е.

Принял:

к.т.н., доцент, Юрова О. В.

к.т.н., Деев М. В.

Пенза 2025

Цель

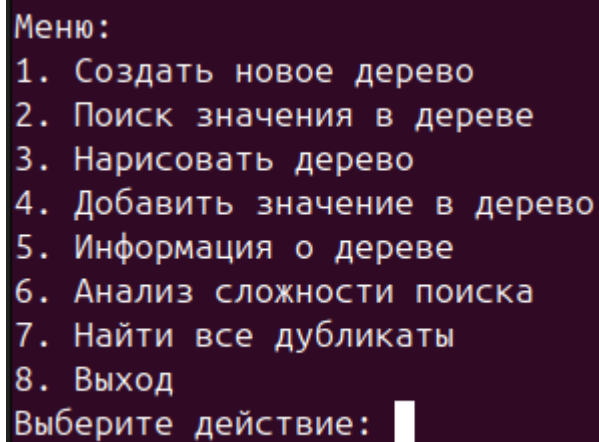
Изучение поиска в бинарном дереве.

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.

Результаты программ



```
Меню:  
1. Создать новое дерево  
2. Поиск значения в дереве  
3. Нарисовать дерево  
4. Добавить значение в дерево  
5. Информация о дереве  
6. Анализ сложности поиска  
7. Найти все дубликаты  
8. Выход  
Выберите действие: █
```

1 Рис. — Меню

```
0. Выход
Выберите действие: 1
=== ЗАПОЛНЕНИЕ ДЕРЕВА ===
Разрешить дубликаты в дереве? (y/n): y
Введите значения для добавления в дерево.
Введите значение: 15
Значение 15 добавлено в дерево.
Добавить еще одно значение? (y/n): y
Введите значение: 10
Значение 10 добавлено в дерево.
Добавить еще одно значение? (y/n): y
Введите значение: 20
Значение 20 добавлено в дерево.
Добавить еще одно значение? (y/n): y
Введите значение: 10
Значение 10 добавлено в дерево.
Добавить еще одно значение? (y/n): y
Введите значение: 5
Значение 5 добавлено в дерево.
Добавить еще одно значение? (y/n): y
Введите значение: 12
Значение 12 добавлено в дерево.
Добавить еще одно значение? (y/n): n
```

2 Рис. — Создание дерева

```
Выберите действие: 2
Введите значение для поиска: 10
✓ Значение 10 найдено в дереве 2 раз(а).

Нажмите Enter для продолжения...
```

3 Рис. — поиск элемента

```
Выберите действие: 3
Дерево:
      20
     /
    15
   /
  12
 /
10
 /
10
  \
   5

Нажмите Enter для продолжения...
```

4 Рис. — Рисунок дерева

```
Выберите действие: 4
Введите значение для добавления: 100
Значение 100 добавлено в дерево.
```

```
Нажмите Enter для продолжения...
```

```
Меню:
```

1. Создать новое дерево
2. Поиск значения в дереве
3. Нарисовать дерево
4. Добавить значение в дерево
5. Информация о дереве
6. Анализ сложности поиска
7. Найти все дубликаты
8. Выход

```
Выберите действие: 3
```

```
Дерево:
```

```
      100
     /  \
    20   \
   /  \   \
  15   12  \
   /  \   \
  10   10  \
   /      \
  5         \
```

5 Рис. — Добавление элемента

```
Выберите действие: 5
```

```
Информация о дереве:
```

- Разрешены дубликаты: Да
- Количество элементов: 7
- Высота дерева: 4

6 Рис. — Анализ дерева

```
=== АНАЛИЗ СЛОЖНОСТИ ПОИСКА ===
```

```
Характеристики дерева:
```

- Количество элементов: 7
- Высота дерева: 4

```
Ожидаемая сложность поиска:  $O(\log n) = O(\log 7) = 3$  операций
```

7 Рис. — Анализ сложности дерева

Вывод:

В ходе выполнения лабораторной работы были разработаны программы для выполнения заданий Лабораторной работы №4 на тему поиска в бинарном дереве.

Листинг

```
#include <iostream>
#include <limits>
#include <cmath>
#include <vector>
#include <map>

using namespace std;

struct TreeNode {
int data;
TreeNode* left;
TreeNode* right;
TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

struct Tree {
TreeNode* root;
bool allowDuplicates;
int size;
Tree() : root(nullptr), allowDuplicates(false), size(0) {}
};

void collectValuesAndLevels(TreeNode* node, int level, map<int,
vector<int>>& valuesMap) {
if (node == nullptr) return;
valuesMap[node->data].push_back(level);
collectValuesAndLevels(node->left, level + 1, valuesMap);
collectValuesAndLevels(node->right, level + 1, valuesMap);
}

void findAllDuplicates(Tree* tree) {
if (tree == nullptr || tree->root == nullptr) {
cout << "Дерево пустое!\n";
return;
}
map<int, vector<int>> valuesMap;
collectValuesAndLevels(tree->root, 0, valuesMap);
bool hasDuplicates = false;
cout << "Найденные дубликаты:\n";
for (const auto& pair : valuesMap) {
if (pair.second.size() > 1) {
hasDuplicates = true;
cout << "Значение " << pair.first << " встречается " <<
pair.second.size() << " раз(а) на уровнях: ";
for (int level : pair.second) {
```

```

cout << level << " ";
}
cout << endl;
}
}
if (!hasDuplicates) {
cout << "Дубликаты не найдены.\n";
}
}
TreeNode* insertWithoutDuplicates(TreeNode* root, int value, int&
size) {
if (root == nullptr) {
size++;
return new TreeNode(value);
}
if (value < root->data) {
root->left = insertWithoutDuplicates(root->left, value, size);
} else if (value > root->data) {
root->right = insertWithoutDuplicates(root->right, value, size);
}
return root;
}
TreeNode* insertWithDuplicates(TreeNode* root, int value, int& size) {
if (root == nullptr) {
size++;
return new TreeNode(value);
}
if (value <= root->data) {
root->left = insertWithDuplicates(root->left, value, size);
} else {
root->right = insertWithDuplicates(root->right, value, size);
}
return root;
}
void waitEnter() {
cout << "\nНажмите Enter для продолжения...";
cin.ignore(numeric_limits<streamsize>::max(), '\n');
cin.get();
}
bool search(TreeNode* root, int target) {
if (root == nullptr) {
return false;
}
if (root->data == target) {
return true;
} else if (target < root->data) {
return search(root->left, target);
} else {
return search(root->right, target);
}
}
}

```

```

int countOccurrences(TreeNode* root, int target) {
    if (root == nullptr) {
        return 0;
    }
    int count = 0;
    if (root->data == target) {
        count = 1;
    }
    return count + countOccurrences(root->left, target) +
        countOccurrences(root->right, target);
}

int getHeight(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }
    int leftHeight = getHeight(root->left);
    int rightHeight = getHeight(root->right);
    return 1 + max(leftHeight, rightHeight);
}

Tree* createTreeFromUserInput() {
    Tree* tree = new Tree();
    int value;
    char choice;
    cout << "=== ЗАПОЛНЕНИЕ ДЕРЕВА ===\n";
    cout << "Разрешить дубликаты в дереве? (y/n): ";
    cin >> choice;
    tree->allowDuplicates = (choice == 'y' || choice == 'Y');
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << "Введите значения для добавления в дерево.\n";
    do {
        cout << "Введите значение: ";
        cin >> value;
        if (cin.fail()) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Ошибка! Пожалуйста, введите целое число.\n";
            continue;
        }
        if (tree->allowDuplicates) {
            tree->root = insertWithDuplicates(tree->root, value, tree->size);
            cout << "Значение " << value << " добавлено в дерево.\n";
        } else {
            if (search(tree->root, value)) {
                cout << "Значение " << value << " уже существует в дереве. Дубликаты не разрешены.\n";
            } else {
                tree->root = insertWithoutDuplicates(tree->root, value, tree->size);
                cout << "Значение " << value << " добавлено в дерево.\n";
            }
        }
    } while (true);
    cout << "Добавить еще одно значение? (y/n): ";
}

```

```

cin >> choice;
cin.ignore(numeric_limits<streamsize>::max(), '\n');
} while (choice == 'y' || choice == 'Y');
return tree;
}
void inOrderTraversal(TreeNode* r, int k) {
if(r == nullptr){
return;
}
inOrderTraversal(r->right, k + 1);
for(int i = 0; i < k; i++){
cout<<" ";
}
cout<<r->data<<"\n";
inOrderTraversal(r->left, k + 1);
}
void deleteTree(TreeNode* root) {
if (root != nullptr) {
deleteTree(root->left);
deleteTree(root->right);
delete root;
}
}
void addValueToTree(Tree* tree) {
if (tree == nullptr) {
cout << "Дерево не создано!\n";
waitForEnter();
return;
}
int value;
cout << "Введите значение для добавления: ";
cin >> value;
if (cin.fail()) {
cin.clear();
cin.ignore(numeric_limits<streamsize>::max(), '\n');
cout << "Ошибка! Пожалуйста, введите целое число.\n";
waitForEnter();
return;
}
if (tree->allowDuplicates) {
tree->root = insertWithDuplicates(tree->root, value, tree->size);
cout << "Значение " << value << " добавлено в дерево.\n";
} else {
if (search(tree->root, value)) {
cout << "Значение " << value << " уже существует в дереве. Дубликаты не разрешены.\n";
} else {
tree->root = insertWithoutDuplicates(tree->root, value, tree->size);
cout << "Значение " << value << " добавлено в дерево.\n";
}
}
}
}

```



```

waitForEnter();
}
void analyzeSearchComplexity(Tree* tree) {
if (tree == nullptr || tree->root == nullptr) {
cout << "Дерево пустое! Сначала создайте дерево.\n";
waitForEnter();
return;
}
cout << "\n=== АНАЛИЗ СЛОЖНОСТИ ПОИСКА ===\n";
int height = getHeight(tree->root);
int n = tree->size;
cout << "Характеристики дерева:\n";
cout << " - Количество элементов: " << n << "\n";
cout << " - Высота дерева: " << height << "\n";
cout << "\nОжидаемая сложность поиска: ";
if (height <= log2(n) + 2) {
int operations = ceil(log2(n));
cout << "O(log n) = O(log " << n << ") = " << operations << "
операций\n";
} else {
cout << "O(n) = O(" << n << ") = " << n << " операций\n";
}
waitForEnter();
}
int main() {
Tree* tree = nullptr;
int choice;
cout << "=== ПРОГРАММА ПОИСКА В ДЕРЕВЕ ===\n";
do {
cout << "\nМеню:\n";
cout << "1. Создать новое дерево\n";
cout << "2. Поиск значения в дереве\n";
cout << "3. Нарисовать дерево\n";
cout << "4. Добавить значение в дерево\n";
cout << "5. Информация о дереве\n";
cout << "6. Анализ сложности поиска\n";
cout << "7. Найти все дубликаты\n";
cout << "8. Выход\n";
cout << "Выберите действие: ";
cin >> choice;
if (cin.fail()) {
cin.clear();
cin.ignore(numeric_limits<streamsize>::max(), '\n');
cout << "Ошибка! Пожалуйста, введите число от 1 до 8.\n";
waitForEnter();
continue;
}
switch (choice) {
case 1: {
if (tree != nullptr) {
deleteTree(tree->root);

```

```

delete tree;
}
tree = createTreeFromUserInput();
break;
}
case 2: {
if (tree == nullptr || tree->root == nullptr) {
cout << "Дерево пустое! Сначала создайте дерево.\n";
} else {
int key;
cout << "Введите значение для поиска: ";
cin >> key;
if (cin.fail()) {
cin.clear();
cin.ignore(numeric_limits<streamsize>::max(), '\n');
cout << "Ошибка! Пожалуйста, введите целое число.\n";
} else {
if (search(tree->root, key)) {
if (tree->allowDuplicates) {
int count = countOccurrences(tree->root, key);
cout << "✓ Значение " << key << " найдено в дереве " << count << "
раз(a).\n";
} else {
cout << "✓ Значение " << key << " найдено в дереве.\n";
}
} else {
cout << "✗ Значение " << key << " не найдено в дереве.\n";
}
}
}
waitForEnter();
break;
}
case 3: {
if (tree == nullptr || tree->root == nullptr) {
cout << "Дерево пустое!\n";
} else {
cout << "Дерево:\n";
inOrderTraversal(tree->root, 0);
cout << endl;
}
waitForEnter();
break;
}
case 4: {
addValueToTree(tree);
break;
}
case 5: {
if (tree == nullptr) {
cout << "Дерево не создано!\n";

```

```

    } else {
        cout << "Информация о дереве:\n";
        cout << " - Разрешены дубликаты: " << (tree->allowDuplicates ? "Да" :
        "Нет") << "\n";
        cout << " - Количество элементов: " << tree->size << "\n";
        if (tree->root != nullptr) {
            cout << " - Высота дерева: " << getHeight(tree->root) << "\n";
        } else {
            cout << " - Дерево пустое\n";
        }
    }
    waitForEnter();
    break;
}
case 6: {
    analyzeSearchComplexity(tree);
    break;
}
case 7: {
    findAllDuplicates(tree);
    waitForEnter();
    break;
}
case 8: {
    cout << "Выход из программы...\n";
    break;
}
default: {
    cout << "Неверный выбор! Пожалуйста, выберите от 1 до 8.\n";
    waitForEnter();
    break;
}
} while (choice != 8);
if (tree != nullptr) {
    deleteTree(tree->root);
    delete tree;
}
return 0;
}

```