Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнили:

студент группы 23ВВВ2

Кокарев Д.С.

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2024

**Название**

Бинарное дерево поиска.

**Цель работы**

Изучить работу бинарного дерева поиска.

**Лабораторное 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.

**Описание метода решения задачи**

*Задание 1 и 2:*

Для поиска значения в уже созданном дереве используется функция FindTree. Она проходит все дерево и при нахождении искомого элемента инкрементирует переменную С (С – это подсчет количества вхождений числа).

*Задание 3:*

Вызывается функция FindTree, изменяющая значение переменной С. С = 1, если текущий элемент дерева совпадает с элементом, который уже встречался ранее.

Функция создания элемента дерева CreateTree вызывается при условии, что С == 0, что исключает создание одинаковых элементов.

*Задание 4:*

Сложность поиска по значению в бинарном дереве определяется с помощью переменной R, которая увеличивается на единицу при каждом вызове функции FindTree. Эта переменная подсчитывает, сколько узлов было посещено во время поиска заданного значения.

Сложность процедуры поиска по значению :

В худшем случае = O(n), так как поиск будет проходить через все узлы;

В среднем случае = O(log n), где n - количество узлов.

**Листинг**

**lab4.cpp**

#include <iostream>

#include <memory>

#include <string>

int C;

int R;

void FindTree(struct Node\* root, int data);

struct Node\* CreateTree(struct Node\* root, struct Node\* r, int data);

int FindLevel(Node\* root, int data, int level = 0);

void print\_tree(struct Node\* r, int l);

Node\* DeleteNode(Node\* root, int data);

Node\* FindMin(Node\* node);

struct Node {

int data;

Node\* left;

Node\* right;

};

int main() {

std::setlocale(LC\_ALL, "");

std::string input;

int D;

Node\* root = nullptr;

std::cout << "Введите (\*) для окончания построения дерева" << std::endl;

while (true) {

std::cout << "Введите число: ";

std::cin >> input;

if (input == "\*") {

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

break;

}

D = std::stoi(input);

C = 0; R = 0;

FindTree(root, D);

if (C == 0) {

root = CreateTree(root, root, D);

}

else {

std::cout << "Введите другое число, отличное от этого" << std::endl;

}

print\_tree(root, 0);

}

std::cout << "Введите (\*) для окончания поиска" << std::endl;

while (true) {

std::cout << "Введите искомое число: ";

std::cin >> input;

if (input == "\*") {

std::cout << "Поиск окончен" << std::endl << std::endl;

break;

}

D = std::stoi(input);

C = 0; R = 0;

FindTree(root, D);

std::cout << "Число вхождений = " << C << std::endl;

std::cout << "Сложность поиска = " << R << std::endl;

int level = FindLevel(root, D);

if (level != -1) {

std::cout << "Уровень элемента " << D << " = " << level << std::endl;

}

else {

std::cout << "Элемент " << D << " не найден в дереве" << std::endl;

}

}

while (true) {

std::cout << "Введите число для удаления (\* для выхода): ";

std::cin >> input;

if (input == "\*") break;

D = std::stoi(input);

root = DeleteNode(root, D);

print\_tree(root, 0);

}

return 0;

}

void FindTree(Node\* root, int data) {

R++;

if (root == nullptr) return;

if (data == root->data) {

C++;

}

if (data < root->data) {

FindTree(root->left, data);

}

else {

FindTree(root->right, data);

}

}

Node\* CreateTree(Node\* root, Node\* r, int data) {

if (r == nullptr) {

r = new Node;

if (r == nullptr) {

std::cerr << "Ошибка выделения памяти" << std::endl;

std::exit(1);

}

r->left = nullptr;

r->right = nullptr;

r->data = data;

if (root == nullptr) return r;

if (data < root->data) root->left = r;

else root->right = r;

return r;

}

if (data < r->data) {

CreateTree(r, r->left, data);

}

else {

CreateTree(r, r->right, data);

}

return root;

}

int FindLevel(Node\* root, int data, int level) {

if (root == nullptr) {

return -1;

}

if (root->data == data) {

return level;

}

int leftLevel = FindLevel(root->left, data, level + 1);

if (leftLevel != -1) {

return leftLevel;

}

int rightLevel = FindLevel(root->right, data, level + 1);

return rightLevel;

}

void print\_tree(Node\* r, int l) {

if (r == nullptr) {

return;

}

print\_tree(r->right, l + 1);

std::cout << l << "-";

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

std::cout << " ";

}

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

print\_tree(r->left, l + 1);

}

Node\* DeleteNode(Node\* root, int data) {

if (root == nullptr) {

return root;

}

if (data < root->data) {

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

}

else if (data > root->data) {

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

}

else {

if (root->left == nullptr) {

Node\* temp = root->right;

delete root;

return temp;

}

else if (root->right == nullptr) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = FindMin(root->right);

root->data = temp->data;

root->right = DeleteNode(root->right, temp->data);

}

return root;

}

Node\* FindMin(Node\* node) {

while (node->left != nullptr) {

node = node->left;

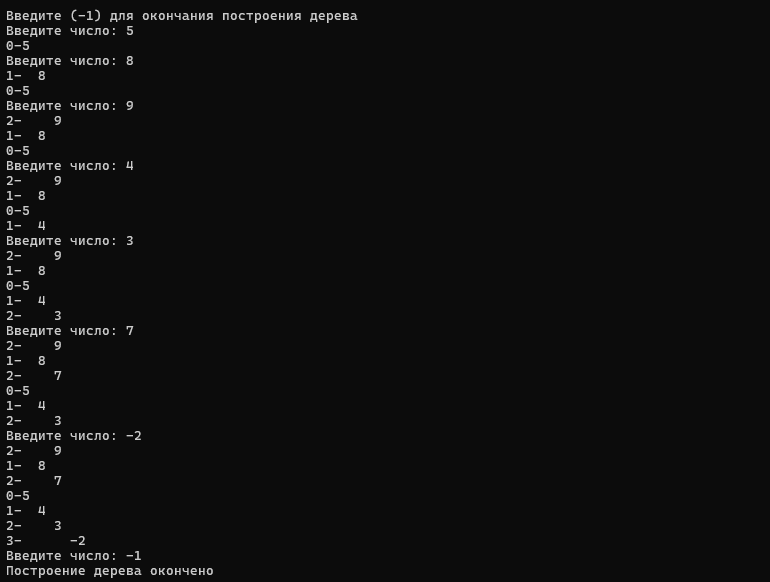
}

return node;

}

**Результаты работы программы**

Результаты работы программы представлены на рисунках:

Рисунок 1 - Создание дерева

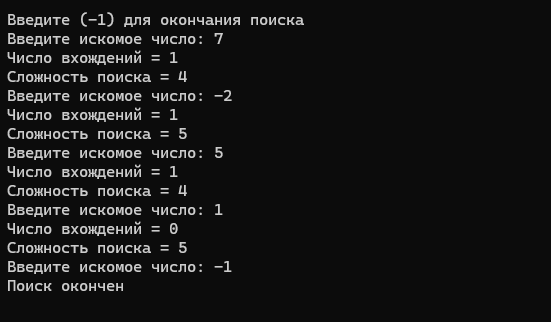


Рисунок 2 - Поиск элементов, число вхождений, сложность поиска

**Вывод**

В ходе выполнения лабораторной работы была разработана программа, выполняющая операции в бинарном дереве поиска. Результаты работы программы совпали с ожидаемыми результатами, следовательно, программа работает без ошибок. Также был получен опыт в создании проектов в среде Microsoft Visual Studio и приобретен навык программирования алгоритмов.