Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра “Вычислительная техника”

Отчет

по лабораторной работе №4

по курсу “ **Логика и основы алгоритмизации инженерных задач**”

на тему “ Бинарное дерево поиска.”

Выполнили студ. группы 24ВВВ4:

Бывшева А.Ю

Конкина С.В

Приняли:

к.т.н. доцент Юрова О.В.

к.э.н. доцент Акифьев И.В.

Пенза 2025

Цель работы: Практическое освоение структуры данных "бинарное дерево поиска" и реализация основных операций над ней. Исследование эффективности алгоритма поиска в зависимости от структуры дерева и анализ временной сложности операций.

Методические указания:

Бинарные деревья – это деревья, у каждого узла которого возможно наличие

только двух сыновей. Двоичные деревья являются упорядоченными.

Двоичное дерево можно представить в виде нелинейного связанного списка.

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными

свойствами:

– значение левого потомка меньше значения родителя;

– значение правого потомка больше значения родителя.

Такие структуры используются для сохранения данных в отсортированном виде.

В простейшем случае для создания элемента списка используется структура, в

которой объединяются полезная информация и ссылка на следующий элемент списка:

struct Node {

int data;

struct Node \*left;

struct Node \*right;

};

В качестве информации в дереве хранятся целые числа.

Обращение к дереву и его элементам осуществляется посредством указателей:

struct Node \*root;

Так как деревья по своей сути являются рекурсивными структурами данных, то и

большинство функций, работающих с деревьями, рекурсивны.

Задание:

1Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном

дереве.

2Реализовать 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.

Задание 1:

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root = NULL;

// функция создания дерева

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

if (r == NULL) {

r = (struct Node\*)malloc(sizeof(struct Node));

r->data = data;

r->left = NULL;

r->right = NULL;

return r;

}

if (data < r->data) {

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

}

else {

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

}

return r;

}

// Реализация алгоритма поиска

struct Node\* search(struct Node\* r, int key) {

if (r == NULL || r->data == key) {

return r;

}

if (key < r->data) {

return search(r->left, key);

}

else {

return search(r->right, key);

}

}

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

if (r == NULL) return;

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

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

printf("%d\n", r->data);

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

}

void free\_tree(struct Node\* r) {

if (r == NULL) return;

free\_tree(r->left);

free\_tree(r->right);

free(r);

}

int main() {

setlocale(LC\_ALL, "");

int D;

root = NULL;

printf("=== задание 1: Алгоритм поиска в бинарном дереве ===\n");

int test\_data[] = { 50, 30, 70, 20, 40, 60, 80 };

int n = 7;

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

root = CreateTree(root, test\_data[i]);

}

printf("Построенное дерево:\n");

print\_tree(root, 0);

// Тестирование поиска

printf("\n--- поиск ---\n");

printf("Введите число для поиска: ");

scanf\_s("%d", &D);

struct Node\* found = search(root, D);

if (found != NULL) {

printf(" Элемент %d найден в дереве!\n", D);

}

else {

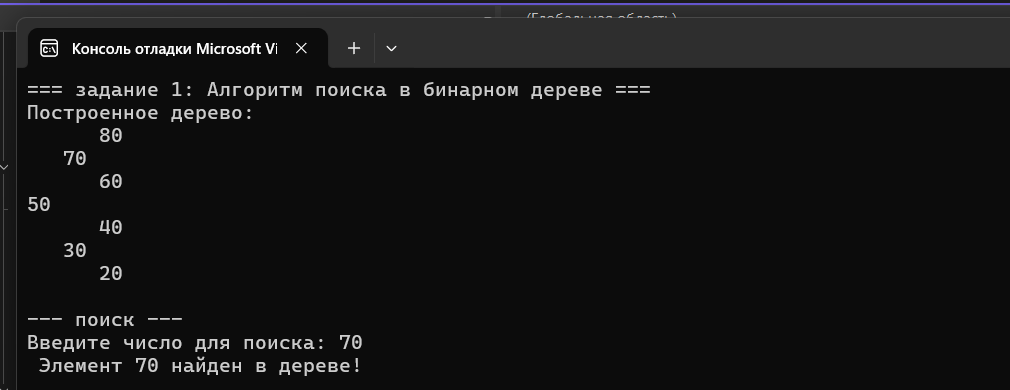
printf(" Элемент %d не найден в дереве.\n", D);

}

free\_tree(root);

return 0;

}

Результат программы:  


Задание 2:

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <ctype.h>

#include <string.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root = NULL;

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

if (r == NULL) {

r = (struct Node\*)malloc(sizeof(struct Node));

r->data = data;

r->left = NULL;

r->right = NULL;

return r;

}

if (data < r->data) {

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

}

else {

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

}

return r;

}

// 2: Функция подсчёта числа вхождений

int count\_occurrences(struct Node\* r, int key) {

if (r == NULL) {

return 0;

}

int count = 0;

if (r->data == key) {

count = 1;

}

// Ищем во всем дереве

count += count\_occurrences(r->left, key);

count += count\_occurrences(r->right, key);

return count;

}

// Вывод дерева

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

if (r == NULL) {

return;

}

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

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

printf(" ");

}

printf("%d\n", r->data);

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

}

void free\_tree(struct Node\* r) {

if (r == NULL) return;

free\_tree(r->left);

free\_tree(r->right);

free(r);

}

// Функция для проверки, состоит ли строка только из цифр и знака минус

int is\_valid\_number(const char\* str) {

int i = 0;

while (isspace((unsigned char)str[i])) {

i++;

}

if (str[i] == '-') {

i++;

}

if (str[i] == '\0') {

return 0;

}

while (str[i] != '\0') {

if (!isdigit((unsigned char)str[i]) && !isspace((unsigned char)str[i])) {

return 0;

}

i++;

}

return 1; // Строка содержит только цифры и пробелы

}

int safe\_input() {

char input[100];

int number;

int valid\_input;

while (1) {

printf("Введите число для подсчёта (-1 для выхода): ");

if (fgets(input, sizeof(input), stdin) == NULL) {

printf("Ошибка ввода. Попробуйте снова.\n");

continue;

}

// Убираем символ новой строки

input[strcspn(input, "\n")] = 0;

// Проверяем, что строка содержит только допустимые символы

if (!is\_valid\_number(input)) {

printf("Ошибка: введите целое число без букв.\n");

continue;

}

// Проверяем, что введено число

valid\_input = sscanf\_s(input, "%d", &number);

if (valid\_input != 1) {

printf("Ошибка: введите целое число.\n");

continue;

}

if (number != -1 && (number < -100000 || number > 100000)) {

printf("Ошибка: число должно быть в диапазоне от -100000 до 100000.\n");

continue;

}

break;

}

return number;

}

int main() {

setlocale(LC\_ALL, "");

int D;

root = NULL;

printf("=== задание 2: Подсчёт числа вхождений элемента ===\n");

int test\_data[] = { 50, 30, 70, 30, 20, 50, 40, 50 };

int n = 8;

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

root = CreateTree(root, test\_data[i]);

}

printf("Построенное дерево:\n");

print\_tree(root, 0);

while (1) {

printf("\n--- подсчет вхождений ---\n");

D = safe\_input();

if (D == -1) {

printf("Выход из программы.\n");

break;

}

int count = count\_occurrences(root, D);

printf("Элемент %d встречается в дереве %d раз(а)\n", D, count);

}

free\_tree(root);

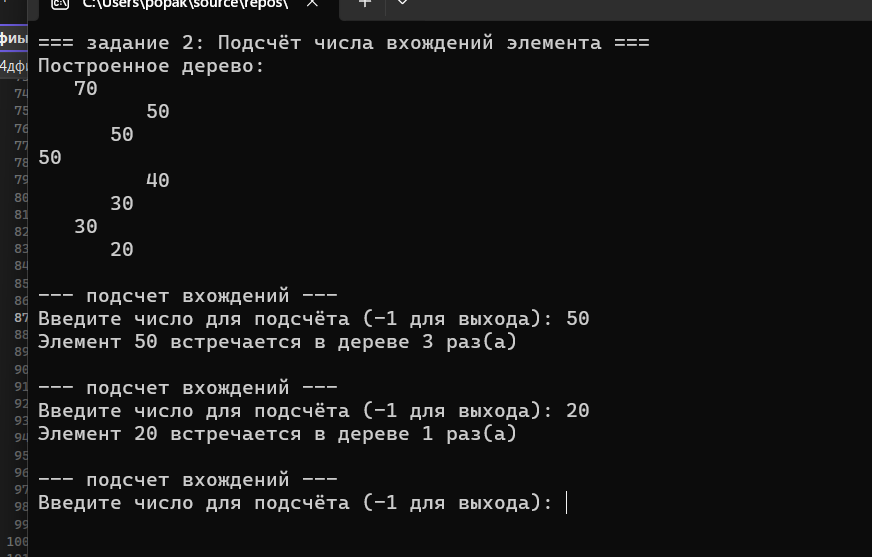
return 0;

}

free\_tree(root);

return 0;

}



Задание 3:

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root = NULL;

// 3: Функция с исключением дубликатов

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

if (r == NULL) {

r = (struct Node\*)malloc(sizeof(struct Node));

r->data = data;

r->left = NULL;

r->right = NULL;

printf(" Добавлен элемент: %d\n", data);

return r;

}

if (data == r->data) {

printf(" Элемент %d уже существует. Пропускаем.\n", data);

return r;

}

if (data < r->data) {

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

}

else {

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

}

return r;

}

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

if (r == NULL) return;

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

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

printf("%d\n", r->data);

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

}

void free\_tree(struct Node\* r) {

if (r == NULL) return;

free\_tree(r->left);

free\_tree(r->right);

free(r);

}

int main() {

setlocale(LC\_ALL, "");

int D;

root = NULL;

printf("=== задание 3: Исключение добавления одинаковых элементов ===\n");

printf("Вводите числа по одному, для завершения введите -1\n\n");

// Интерактивное создание дерева

while (1) {

printf("Введите число: ");

if (scanf\_s("%d", &D) != 1) {

printf("Ошибка ввода! Введите целое число.\n");

while (getchar() != '\n');

continue;

}

if (D == -1) {

printf("Построение дерева окончено\n\n");

break;

}

root = CreateTree(root, D);

}

printf("\nИтоговое дерево (без дубликатов):\n");

print\_tree(root, 0);

printf("\n--- проверка ---\n");

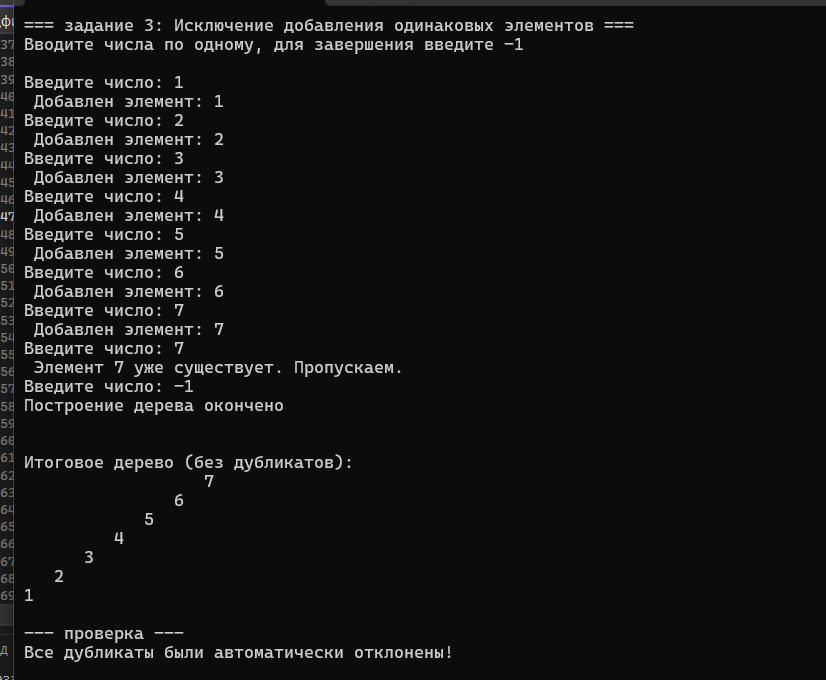
printf("Все дубликаты были автоматически отклонены!\n");

free\_tree(root);

return 0;

}

Результат программы:



Задание 4:

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <math.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root = NULL;

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

if (r == NULL) {

r = (struct Node\*)malloc(sizeof(struct Node));

r->data = data;

r->left = NULL;

r->right = NULL;

return r;

}

if (data < r->data) {

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

}

else {

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

}

return r;

}

// Функция поиска для демонстрации

struct Node\* search(struct Node\* r, int key, int\* steps) {

(\*steps)++;

if (r == NULL || r->data == key) {

return r;

}

if (key < r->data) {

return search(r->left, key, steps);

}

else {

return search(r->right, key, steps);

}

}

// Функция для вычисления высоты дерева

int tree\_height(struct Node\* r) {

if (r == NULL) return 0;

int left\_height = tree\_height(r->left);

int right\_height = tree\_height(r->right);

return 1 + (left\_height > right\_height ? left\_height : right\_height);

}

int count\_nodes(struct Node\* r) {

if (r == NULL) return 0;

return 1 + count\_nodes(r->left) + count\_nodes(r->right);

}

// Функция вывода дерева

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

if (r == NULL) {

return;

}

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

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

printf(" ");

}

printf("%d\n", r->data);

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

}

void free\_tree(struct Node\* r) {

if (r == NULL) return;

free\_tree(r->left);

free\_tree(r->right);

free(r);

}

int main() {

setlocale(LC\_ALL, "");

int D;

root = NULL;

printf("=== задание 4: Оценка сложности процедуры поиска ===\n\n");

// Ручной ввод дерева

printf("Вводите числа для построения дерева (для завершения введите -1):\n");

while (1) {

printf("Введите число: ");

scanf\_s("%d", &D);

if (D == -1) {

printf("Построение дерева завершено.\n\n");

break;

}

root = CreateTree(root, D);

}

// Вывод построенного дерева

printf("Построенное дерево:\n");

print\_tree(root, 0);

printf("\n");

int height = tree\_height(root);

int node\_count = count\_nodes(root);

printf("АНАЛИЗ СЛОЖНОСТИ ПОИСКА\n");

printf("==========================\n");

printf("Количество элементов (n): %d\n", node\_count);

printf("Высота дерева (h): %d\n", height);

printf("Теоретический log₂(n) ≈ %.2f\n", log2(node\_count));

printf("\n");

if (height <= log2(node\_count) + 2) {

printf("• Ваше дерево близко к сбалансированному \n");

}

else {

printf("• Ваше дерево ближе к вырожденному \n");

}

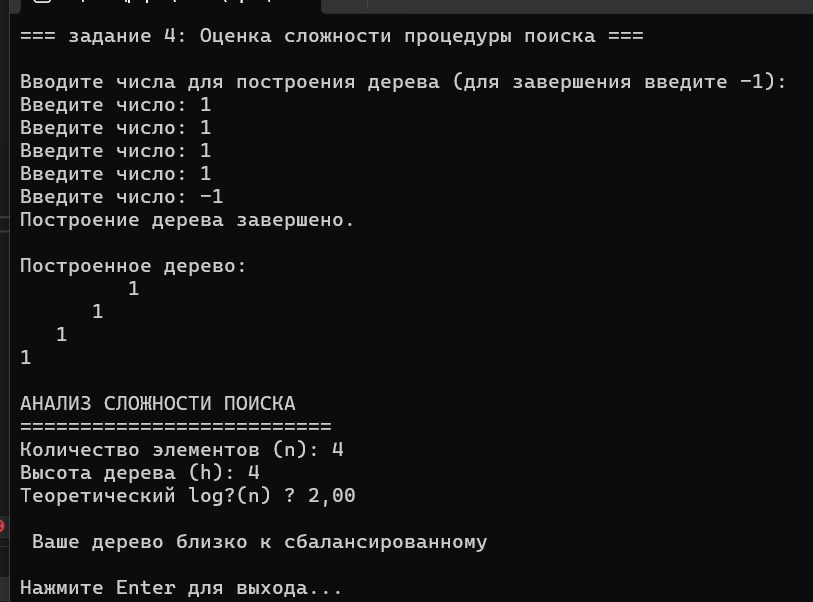
free\_tree(root);

printf("\nНажмите Enter для выхода...");

getchar(); getchar();

return 0;

}

Результат программы:  


Вывод: В ходе выполнения лабораторной работы были успешно реализованы все основные операции с бинарным деревом поиска. Создана программа, позволяющая добавлять элементы в дерево с соблюдением свойств BST, осуществлять поиск элементов, подсчитывать количество вхождений и исключать дубликаты при добавлении.