Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №4
по курсу «Программирование»
на тему «Бинарное дерево поиска»

Выполнили:
студенты группы 24ВВВ4
Суходолов И.А.
Чернышевский Е.И.
Принял:
к. т. н., доцент Юрова О.В.
к. э. н., доцент Акифьев И.В.

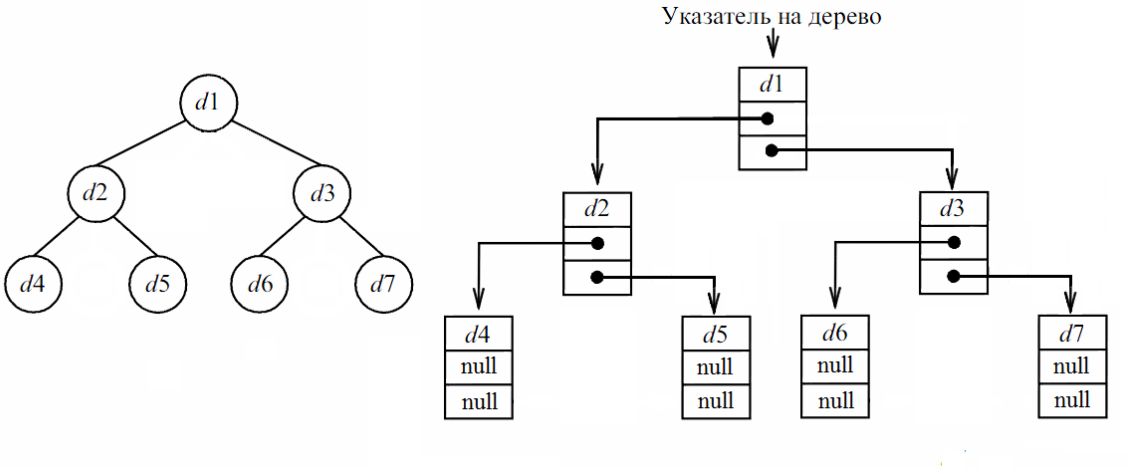
Пенза 2025

Цель работы:

Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве, а также сделать функцию подсчёта чисел введённый в заданный элемент в дереве и изменить функцию добавления элементов для исключения добавления одинаковых символов.

Общие сведения.

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.

Практическая часть

<u>Задание 1:</u>

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.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

// Структура узла бинарного дерева
struct Node {
    int data;
```

```
    struct Node* left;
    struct Node* right;
};

struct Node* root = NULL;

// Функция создания нового узла
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Ошибка выделения памяти\n");
        exit(1);
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Функция добавления элемента в дерево
struct Node* addElement(struct Node* node, int data) {
    if (node == NULL) {
        return createNode(data);
    }

    if (data == node->data) {
        printf("Элемент %d уже существует в дереве. Пропускаем.\n", data);
        return node;
    }

    if (data < node->data) {
        node->left = addElement(node->left, data);   // Меньшие значения - в ЛЕВОЕ поддерево
    }
    else {
        node->right = addElement(node->right, data); // Большие значения - в ПРАВОЕ поддерево
    }

    return node;
}

// Функция поиска элемента в дереве
struct Node* searchElement(struct Node* node, int key) {
    if (node == NULL || node->data == key) {
        return node;
    }

    if (key < node->data) {
        return searchElement(node->left, key);
    }
    else {
        return searchElement(node->right, key);
    }
}

// Функция подсчета вхождений элемента
int countOccurrences(struct Node* node, int key) {
    if (node == NULL) {
        return 0;
    }

    if (key < node->data) {
        return countOccurrences(node->left, key);   // Ищем в ЛЕВОМ поддереве
    }
    else if (key > node->data) {
        return countOccurrences(node->right, key);  // Ищем в ПРАВОМ поддереве
    }
```

```
    else {
        return 1 + countOccurrences(node->left, key) + countOccurrences(node->right, key);
    }
}

// Функция симметричного вывода дерева (in-order traversal)
void printInOrder(struct Node* node) {
    if (node != NULL) {
        printInOrder(node->left);
        printf("%d ", node->data);
        printInOrder(node->right);
    }
}

// Функция красивого вывода дерева (вертикальный вывод)
void printTreeVertical(struct Node* node, int level) {
    if (node == NULL) {
        return;
    }
    printTreeVertical(node->right, level + 1);

    for (int i = 0; i < level; i++) {
        printf("    ");
    }

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

    printTreeVertical(node->left, level + 1);
}

// Вспомогательная функция для красивого вывода результата поиска
void printSearchResult(int key) {
    struct Node* result = searchElement(root, key);
    if (result != NULL) {
        printf("Элемент %d найден в дереве!\n", key);
    }
    else {
        printf("Элемент %d не найден в дереве.\n", key);
    }
}

// Вспомогательная функция для вывода статистики
void printStatistics(int key) {
    int count = countOccurrences(root, key);
    printf("Элемент %d встречается в дереве %d раз(а)\n", key, count);
}

// Очистка дерева
void freeTree(struct Node* node) {
    if (node != NULL) {
        freeTree(node->left);
        freeTree(node->right);
        free(node);
    }
}

int main() {
    setlocale(LC_ALL, "");

    int inputValue;
    int searchValue;

    printf("=== ПОСТРОЕНИЕ БИНАРНОГО ДЕРЕВА ПОИСКА ===\n");
    printf("Введите числа для построения дерева\n");
    printf("-1 - окончание ввода\n\n");
```

```
    while (1) {
        printf("Введите число: ");
        scanf("%d", &inputValue);

        if (inputValue == -1) {
            printf("Построение дерева завершено.\n\n");
            break;
        }

        root = addElement(root, inputValue);
    }

    // Вывод построенного дерева
    printf("\n=== ПОСТРОЕННОЕ ДЕРЕВО ===\n");
    printf("Симметричный вывод: ");
    printInOrder(root);
    printf("\n\nВертикальное представление:\n");
    printTreeVertical(root, 0);

    // Поиск элементов
    printf("\n=== ПОИСК ЭЛЕМЕНТОВ В ДЕРЕВЕ ===\n");
    printf("Введите значение для поиска: ");
    scanf("%d", &searchValue);
    printSearchResult(searchValue);

    // Подсчет вхождений
    printf("\n=== ПОДСЧЕТ ВХОЖДЕНИЙ ЭЛЕМЕНТА ===\n");
    printf("Введите значение для подсчета: ");
    scanf("%d", &searchValue);
    printStatistics(searchValue);

    // Дополнительный интерактивный поиск
    printf("\n=== ДОПОЛНИТЕЛЬНЫЙ ПОИСК ===\n");
    printf("Вводите значения для поиска (-1 для выхода):\n");

    while (1) {
        printf("Введите значение для поиска: ");
        scanf("%d", &searchValue);

        if (searchValue == -1) {
            break;
        }

        printSearchResult(searchValue);
        printStatistics(searchValue);
        printf("\n");
    }

    // Освобождение памяти
    freeTree(root);
    root = NULL;

    printf("Программа завершена.\n");
    return 0;
}
```

Вывод:

Реализовали алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве, а также сделали функцию подсчёта чисел введённый в заданный элемент в дереве и изменили функцию добавления элементов для исключения добавления одинаковых символов.