Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска**.**»

Выполнили:

студенты группы 23ВВВ4

Брагин А.М.

Зарубин Я.

Герасимов К.

Приняли:

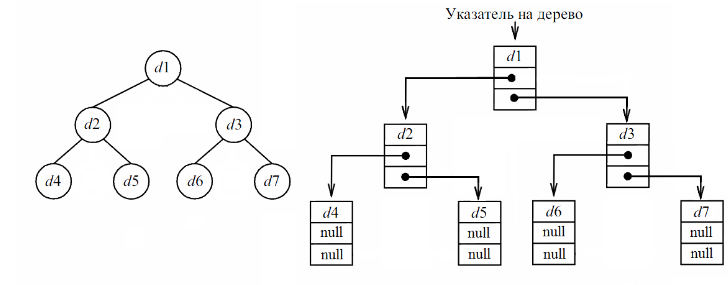
Деев М.В.

Юрова О.В.

Пенза 2024

**Общие сведения.**

Бинарные деревья – это деревья, у каждого узла которого возможно наличие только двух сыновей. Двоичные деревья являются упорядоченными.



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.

Функция создания дерева выделяет память под каждый новый элемент и добавляет его в дерево:

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

{

if (r == NULL)

{

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

if (r == NULL)

{

printf("Ошибка выделения памяти");

exit(0);

}

r->left = NULL;

r->right = NULL;

r->data = data;

if (root == NULL) 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;

}

Функция вывода дерева на экран (дерево выводится повёрнутым на 90 градусов, корень находится слева):

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);

}

Программа, использующая приведенные функции:

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1;

root = NULL;

printf("-1 - окончание построения дерева\n");

while (start)

{

printf("Введите число: ");

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

if (D == -1)

{

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

start = 0;

}

else

root = CreateTree(root, root, D);

}

print\_tree(root,0);

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

return 0;

}

**Практическая часть:**

**Задание**

1. Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.
2. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

Листинг программы

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Структура узла бинарного дерева

struct Node {

int data; // Данные узла

struct Node\* left; // Указатель на левое поддерево

struct Node\* right; // Указатель на правое поддерево

};

// Функция создания нового узла и добавления его в дерево

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

// Если дерево пусто, создаем новый узел

if (r == NULL) {

r = (struct Node\*)malloc(sizeof(struct Node)); // Выделяем память под новый узел

if (r == NULL) {

printf("Ошибка выделения памяти\n");

exit(0); // Завершаем программу при ошибке выделения памяти

}

r->left = NULL; // Левое поддерево пусто

r->right = NULL; // Правое поддерево пусто

r->data = data; // Присваиваем значение новому узлу

return r; // Возвращаем указатель на новый узел

}

// Если значение больше текущего узла, идем в правое поддерево

if (data > r->data) {

r->right = CreateTree(r->right, data); // Рекурсивный вызов для правого поддерева

}

// Если значение меньше, идем в левое поддерево

else if (data < r->data) {

r->left = CreateTree(r->left, data); // Рекурсивный вызов для левого поддерева

}

return r; // Возвращаем корень дерева

}

// Функция вычисления высоты дерева (максимальная глубина дерева)

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

if (root == NULL)

return 0; // Высота пустого дерева равна 0

// Рекурсивное вычисление высоты левого и правого поддерева

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

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

// Возвращаем большую из двух высот, прибавив 1 (для текущего узла)

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

}

// Функция для вывода пробелов

void print\_spaces(int count) {

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

printf(" "); // Выводим указанное количество пробелов

}

}

// Функция печати узлов дерева на заданном уровне

void print\_current\_level(struct Node\* root, int level, int indent\_space, int node\_space) {

// Если узел пустой, печатаем пробелы вместо него

if (root == NULL) {

print\_spaces(indent\_space + node\_space); // Печатаем пробелы для отсутствующего узла

return;

}

// Если уровень равен 1, печатаем данные узла

if (level == 1) {

printf("%d", root->data); // Печатаем значение узла

print\_spaces(node\_space); // Печатаем пробелы между узлами

}

// Иначе продолжаем рекурсивный вызов для левого и правого поддерева

else if (level > 1) {

print\_current\_level(root->left, level - 1, indent\_space / 2, node\_space); // Левое поддерево

print\_current\_level(root->right, level - 1, indent\_space / 2, node\_space); // Правое поддерево

}

}

// Функция печати дерева уровнями с отступами для симметрии

void print\_tree(struct Node\* root) {

int h = tree\_height(root); // Вычисляем высоту дерева

int node\_space; // Переменная для хранения расстояния между узлами

// Цикл по уровням дерева (от 1 до h)

for (int i = 1; i <= h; i++) {

// Задаем начальный отступ для корня (4 пробела) и для остальных уровней (0)

int indent\_space = (i == 1) ? 4 : 0;

// Задаем расстояние между узлами для каждого уровня

if (i == 1) {

node\_space = 0; // Для корня пробелы не нужны

}

else if (i == 2) {

node\_space = 7; // Для второго уровня расстояние между узлами 7 пробелов

}

else {

node\_space = 3; // Для третьего уровня расстояние между узлами 3 пробела

}

// Печатаем отступы для текущего уровня

print\_spaces(indent\_space);

// Печатаем узлы текущего уровня

print\_current\_level(root, i, indent\_space, node\_space);

printf("\n"); // Переход на новую строку после печати уровня

}

}

int main() {

struct Node\* root = NULL; // Инициализация корня дерева

int D, start = 1; // Переменные для ввода данных

// Ввод чисел для построения дерева

printf("-1 - окончание построения дерева\n");

while (start) {

printf("Введите число: ");

scanf("%d", &D); // Чтение числа

if (D == -1) { // Если введено -1, завершить ввод

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

start = 0; // Устанавливаем флаг завершения

}

else {

root = CreateTree(root, D); // Добавляем число в дерево

}

}

// Печать дерева по уровням

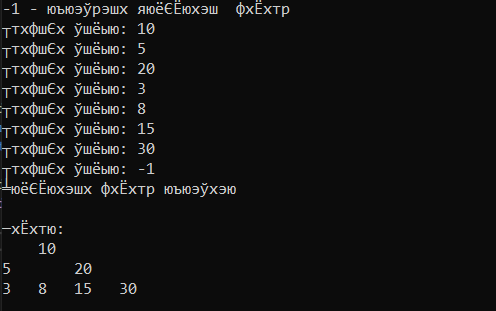
printf("Дерево:\n");

print\_tree(root);

return 0; // Завершение программы

}

Результат работы программы:



### Выводы