МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №4  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Оценка времени выполнения программ»

Выполнил:

Студент группы 23ВВВ2

Чупраков С.В.

Приняли:

Митрохин М. А.  
Юрова О.В.

Пенза 2024

**Цель работы**

Разработка и реализация структур данных, таких как бинарное дерево поиска, с использованием базовых принципов программирования. Создание функций для работы с бинарным деревом поиска.

**Задание**

**Задание 1:**

Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.

**Задание 2:**

Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

**Задание 3:**

Изменить функцию добавления элементов для исключения добавления одинаковых символов.

**Задание 4:**

Оценить сложность процедуры поиска по значению в бинарном дереве.

**Ход работы**

**Задание 1.**

С использованием заранее приведённых функций создали функцию поиска find(). Функция использует особенности сортировки бинарного дерево поиска для поиска нужного элемента.

void find(struct Node\* r, int find\_data) {

if (r->data == find\_data) {

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

return;

}

else if(r->data < find\_data && r->left!=NULL){

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

find(r->left, find\_data);

}

else if(r->data > find\_data && r->right != NULL)

{

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

find(r->right, find\_data);

}

else {

printf("Тут его нет :(");

return;

}

}

Вызов функции в main().

printf("Введите число которое необходимо найти: ");

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

find(root, D);

**Задание 2.**

На основе функции поиска реализована функция подсчёта числа вхождений заданного элемента в дерево.

int find\_input(struct Node\* r, int find\_data, int count\_fun, int m) {

if (r->data == find\_data) {

count\_fun += 1;

}

if (m < count\_fun) {

m=count\_fun;

}

if( r->left!=NULL){

m=find\_input(r->left, find\_data, count\_fun, m);

}

if(r->right != NULL)

{

m=find\_input(r->right, find\_data, count\_fun, m);

}

return m;

}

Вызов функции в main().

printf("Введите число которое необходимо найти: ");

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

count=find\_input(root, D, count, 0);

printf("Элемент встретился %d раз", count);

**Задание 3.**

Чтобы сделать проверку для исключения добавления одинаковых элементов использован алгоритм работы функции find(). Далее эта функция вызывается при каждом добавлении.

int find(struct Node\* r, int find\_data) {

int f = 0;

if (r->data == find\_data) {

f=1;

}

else if (r->data < find\_data && r->left != NULL) {

f=find(r->left, find\_data);

}

else if (r->data > find\_data && r->right != NULL)

{

f=find(r->right, find\_data);

}

else {

f=0;

}

return f;

}

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

{

int m = 0;

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;

}

m = find(r, data);

if (m == 0) {

if (data > r->data )

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

else

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

}

else {

printf("Элемент уже есть\n");

}

return root;

}

**Задание 4.**

Сложность алгоритма поиска будет зависеть от высоты дерева.

В худшем случае, когда дерево вырождено в линейную структуру, например все узлы имеют только одного потомка. Тогда высота дерева будет равна количеству узлов и сложность будет равна **О(n),** n- количество узлов.

Если дерево сбалансировано, то высота дерева будет логарифмической, а сложность будет равна **О(log(n))**

### Результаты работы программы

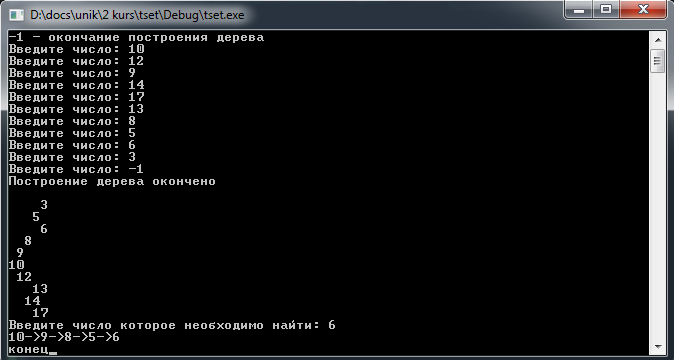


Рисунок 1 — Результаты работы программы пункта №1

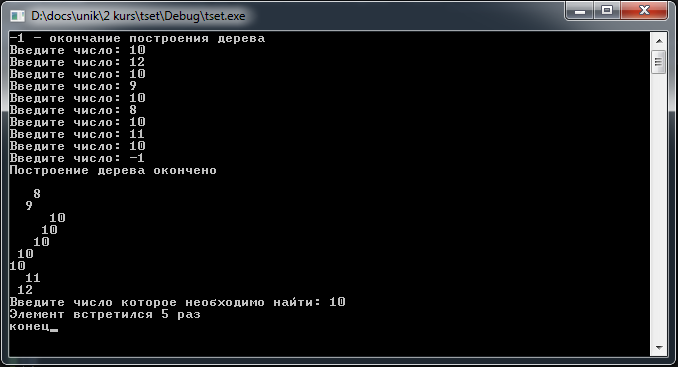


Рисунок 2 — Результаты работы программы пункта №2

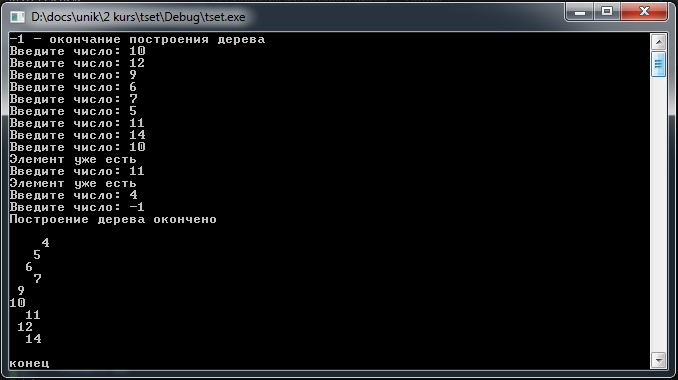
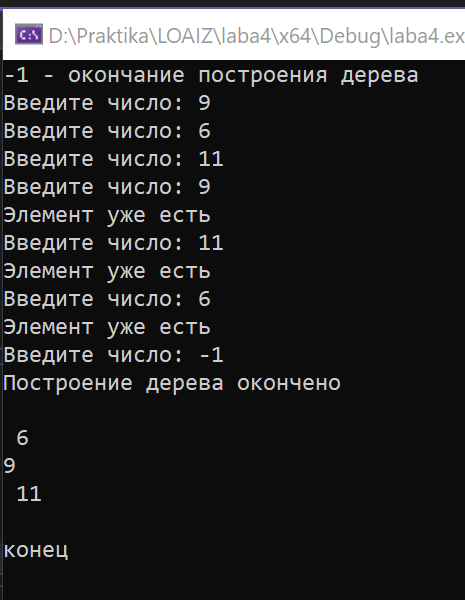


Рисунок 3 — Результаты работы программы пункта №3

**Рисунок 4 — Результаты работы программы пункта №3**

### Вывод

В результате выполнения работы были успешно разработаны и реализованы функции работы с бинарным деревом поиска: поиск заданного элемента, подсчет числа вхождения заданного элемента в дерево и дополнена функция создания элемента дерева для исключения повторного добавления элемента.

**Листинг**

**Задание 1**

﻿#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root;

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;

}

void find(struct Node\* r, int find\_data) {

if (r->data == find\_data) {

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

return;

}

else if(r->data < find\_data && r->left!=NULL){

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

find(r->left, find\_data);

}

else if(r->data > find\_data && r->right != NULL)

{

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

find(r->right, find\_data);

}

else {

printf("Тут его нет :(");

return;

}

}

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);

}

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);

printf("Введите число которое необходимо найти: ");

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

find(root, D);

printf("\nконец");

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

return 0;

}

**Задание 2**

﻿#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root;

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;

}

int find\_input(struct Node\* r, int find\_data, int count\_fun, int m) {

if (r->data == find\_data) {

count\_fun += 1;

}

if (m < count\_fun) {

m=count\_fun;

}

if( r->left!=NULL){

m=find\_input(r->left, find\_data, count\_fun, m);

}

if(r->right != NULL)

{

m=find\_input(r->right, find\_data, count\_fun, m);

}

return m;

}

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);

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1, count=0;

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);

printf("Введите число которое необходимо найти: ");

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

count=find\_input(root, D, count, 0);

printf("Элемент встретился %d раз", count);

printf("\nконец");

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

return 0;

}

**Задание 3**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root;

int find(struct Node\* r, int find\_data) {

int f = 0;

if (r->data == find\_data) {

f = 1;

}

else if (r->data < find\_data && r->left != NULL) {

f = find(r->left, find\_data);

}

else if (r->data > find\_data && r->right != NULL)

{

f = find(r->right, find\_data);

}

else {

f = 0;

}

return f;

}

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

{

int m = 0;

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 (r->data == data) { m = 1; }

if (m == 0) {

if (data > r->data)

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

else

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

}

else {

printf("Элемент уже есть\n");

m = 0;

}

return root;

}

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);

}

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);

printf("\nконец");

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

return 0;

}