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 «Бинарное дерево поиска»

Выполнили:

Студенты группы 23ВВВ2

Лисов Е.А.

Кочегин В.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 свойства бинарного дерева для эффективного поиска нужного элемента.

struct Node\* find(struct Node\* root, int data) {

if (root == NULL) {

return NULL;

}

if (data == root->data) {

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

return root;

}

if (data > root->data) {

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

return find(root->left, data);

}

else {

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

return find(root->right, data);

}

}

**Задание 2.**

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

int find(struct Node\* root, int data, int\* count) {

if (root == NULL) {

return NULL;

}

if (data == root->data) {

(count)++;

}

if (data > root->data) {

find(root->left, data, count);

}

else {

find(root->right, data, count);

}

}

**Задание 3.**

Для реализации проверки на исключения добавления одинаковых элементов

Была изменена функция CreateTree(), которая использует для проверки алгоритм работы функции find() из 1 задания.

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;

}

struct Node\* f = find(root, data);

if (f == NULL)

{

if (data > r->data)

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

else

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

}

else printf("это эл повторяется.\n");

return root;

}

**Задание 4.**

Сложность процедуры поиска по значению в бинарном дереве составляет **O(log2(n))**. Это связано с тем, что на каждом уровне дерева количество узлов уменьшается вдвое, что позволяет быстро находить искомый элемент.

### Результаты работы программы

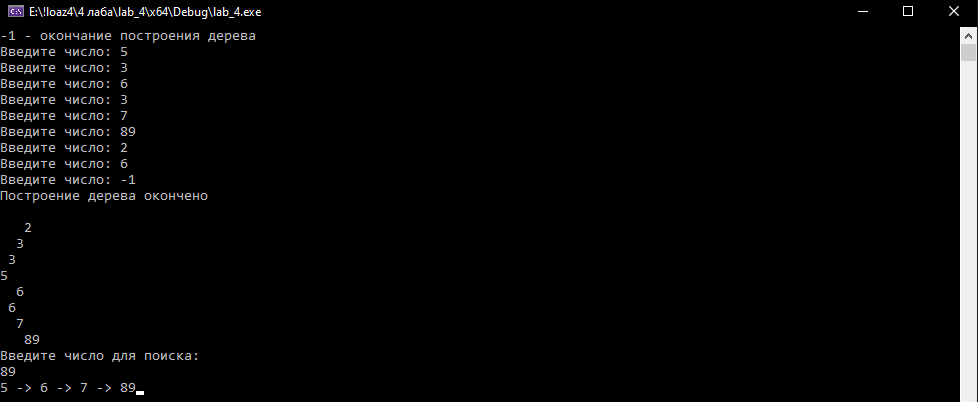


Рисунок 1 — Результаты работы программы пункта №1



Рисунок 2 — Результаты работы программы пункта №2



Рисунок 3 — 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 "string.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 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);

}

//задание 1

struct Node\* find(struct Node\* root, int data) {

if (root == NULL) {

return NULL;

}

if (data == root->data) {

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

return root;

}

if (data > root->data) {

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

return find(root->left, data);

}

else {

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

return find(root->right, data);

}

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1, poisk;

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 для поиска: \n");

scanf("%d", &poisk);

find(root, poisk);

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

return 0;

}

**Задание 2:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdio.h"

#include "stdlib.h"

#include "string.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 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 find(struct Node\* root, int data, int count) {

if (root == NULL) {

return count;

}

if (data == root->data) {

(count)++;

}

if (data > root->data) {

find(root->left, data, count);

}

else {

find(root->right, data, count);

}

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1, poisk, 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 для поиска: \n");

scanf("%d", &poisk);

count= find(root, poisk, count);

if (count != 0) {

printf("эл %d найден, его количество: %d", poisk, count);

}

else {

printf("такого нет.......");

}

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

return 0;

}

**Задание 3:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include "locale.h"

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root;

struct Node\* prov;

struct Node\* find(struct Node\* root, int data) {

if (root == NULL) {

return NULL;

}

if (data == root->data) {

return root;

}

if (data > root->data) {

return find(root->left, data);

}

else {

return find(root->right, data);

}

}

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;

}

struct Node\* f = find(root, data);

if (f == NULL)

{

if (data > r->data)

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

else

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

}

else printf("это эл повторяется.\n");

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, poisk;

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;

}