Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнила:

студентка группы 22ВВВ2

Расторгуева К.В.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Бинарное дерево поиска.

**Цель работы**

Изучение бинарных деревьев поиска. Написание основных функций для работы с бинарными деревьями поиска.

**Лабораторное задание**

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.
4. \* Оценить сложность процедуры поиска по значению в бинарном дереве.

**Пояснительный текст к программе**

Задание 1-2:

Программа содержит основные функции работы со списком: создание бинарного дерева, добавление элемента, просмотр бинарного дерева, нахождение нужного элемента в дереве. Функция поиска принимает искомое значение, которое сравнивается с каждым значением узла дерева. При совпадении значений увеличивается счетчик *count*.

int find(struct Node\* r, int D\_find) {

if (r == NULL){

return count;

}

if (r->data == D\_find) count++;

find(r->right, D\_find);

find(r->left, D\_find);

}

Задания 3:

В функции *CreateTree()* задаётся дополнительное условие: новый элемент сравнивается с каждым узлом на пути своего добавления. Если элементы равны, то возвращается корень дерева, иначе продолжается операция добавления.

if (data == r->data) {

printf("Элемент уже существует\n");

return root;

}

else {

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

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

}

Назначение отдельных операторов программы указывается в виде комментариев. Программа завершает свою работу после нажатия на любую клавишу на клавиатуре.

**Результат выполнения программы**

Результат работы программы для заданий 1 и 2 представлены на рис. 1-3.

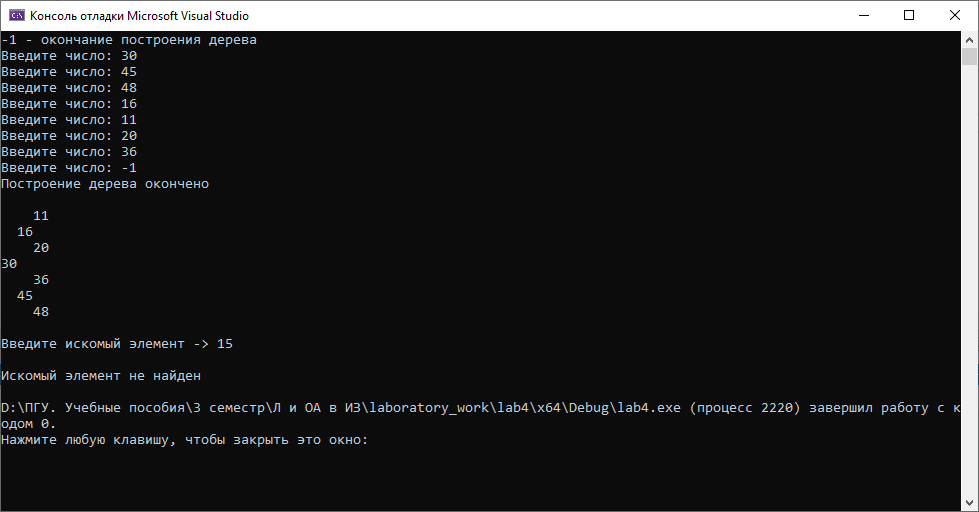


Рисунок 1 – поиск заданного элемента

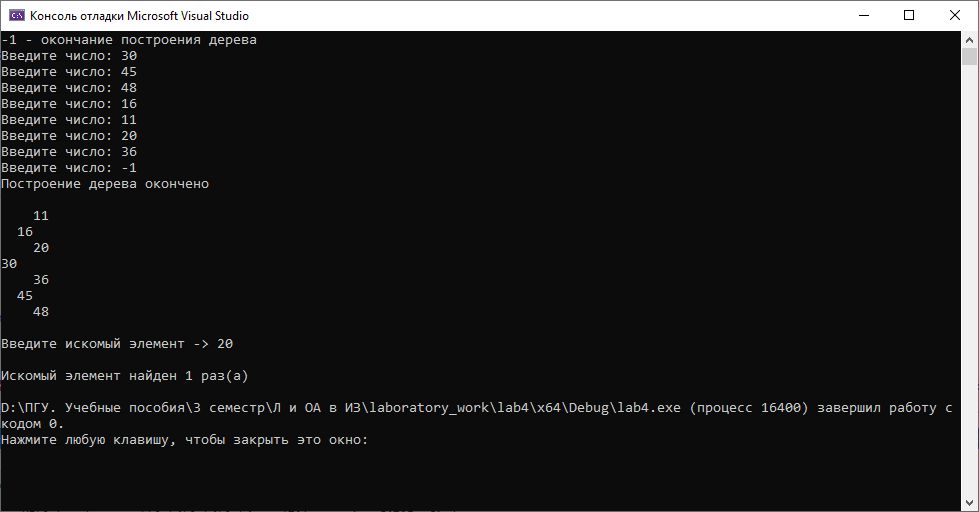


Рисунок 2 – поиск заданного элемента

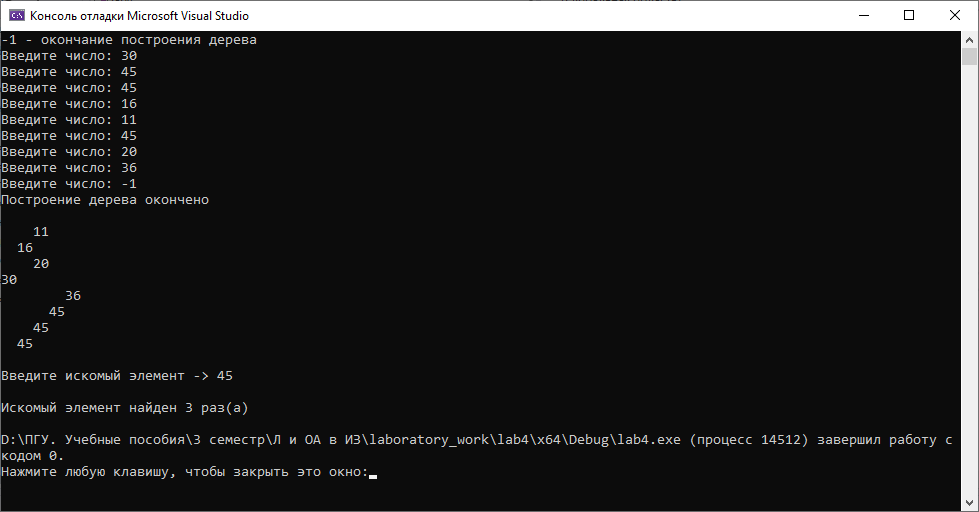


Рисунок 3 – поиск заданного элемента

Результат работы программы для задания 3 представлены на рис. 4.

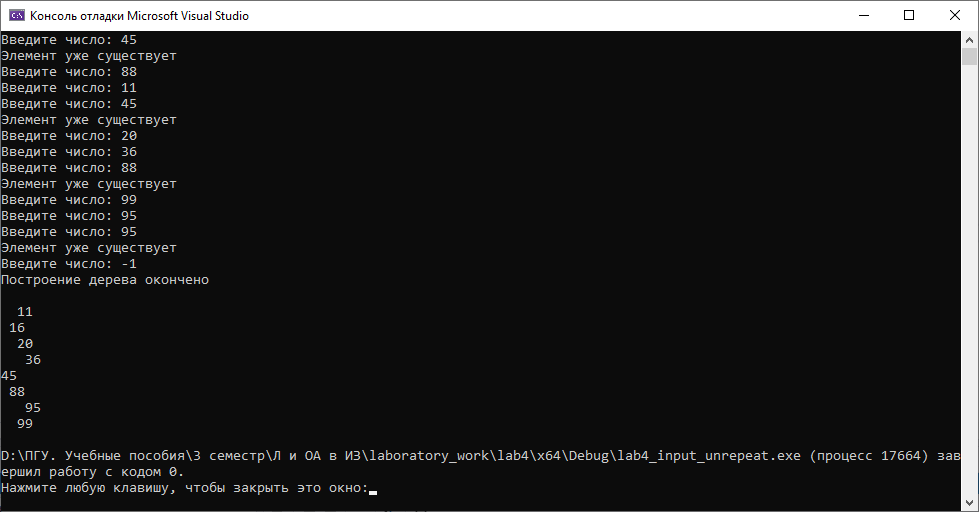


Рисунок 4 – добавление элементов без повторений

Результаты вычисления вручную совпали с вычислениями программы.

**Оценка сложности**

Сложность процедуры поиска по значению в бинарном дереве

* в среднем = О(log n)
* в худшем случае = O(n)

**Выводы**

В ходе выполнения лабораторной работы была разработана на языке Си программа, осуществляющая работу с бинарным деревом поиска:

– реализован алгоритм поиска значения в бинарном дереве;

– реализована функция подсчёта числа вхождений заданного элемента в бинарное дерево;

– реализована функция, исключающая добавление одинаковых символов в бинарное дерево.

Результаты работы программы совпали с результатами расчета вручную.

**Приложение**

**Программа**

Задания 1-2.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root;

int count = 0;

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\* r, int D\_find) {

if (r == NULL){

return count;

}

if (r->data == D\_find) count++;

find(r->right, D\_find);

find(r->left, D\_find);

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1;

int D\_find;

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("\nВведите искомый элемент -> ");

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

count = find(root, D\_find);

if (count) printf("\nИскомый элемент найден %d раз(а)\n", count);

else printf("\nИскомый элемент не найден\n");

return 0;

}

Задание 3.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.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) {

printf("Элемент уже существует\n");

return root;

}

else {

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 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);

return 0;

}