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.

В начале программы определяется структура *Node*, состоящая из информационного поля data и указателей на правую и левую ветки бинарного дерева.

struct Node {

int data; //полезная информация

struct Node\* left; //левая ветка

struct Node\* right; //правая ветка

};

Также объявляется указатель на корень дерева и счетчик для подсчета числа вхождений заданного элемента в дерево

struct Node\* root; //указатель на дерево

int count = 0; //подсчёт числа вхождений заданного элемента в дерево

Рекурсивная функция создания дерева *CreateTree()* создаёт новый узел бинарного дерева. Для этого необходимо дойти до конца дерева, где узел *r* будет принимать значение *NULL*. Если это условие выполняется, то выделяется память для нового узла с помощью функции *malloc().* Для созданного узла создаются указатели на следующие элементы: указатели на левую и правую ветку ссылаются на *NULL*. Числовое значение, переданное в параметрах функции, записывается в информационное поле *data* структуры *Node*.

if (r == NULL) //конец дерева, создание нового узла

{

r = (struct Node\*)malloc(sizeof(struct Node)); //выделение памяти для нового элемента

if (r == NULL) //память не выделена, завершение программы

{

printf("Ошибка выделения памяти");

exit(0);

}

r->left = NULL; //для текщего узла ссылка на левую ветку не определена, приравниваем к NULL

r->right = NULL; //для текщего узла ссылка на правую ветку не определена, приравниваем к NULL

r->data = data; //в поле data заносится числовое значение

Отдельно обрабатывается случай, когда дерево ещё не было создано до этого. Если корень дерева не создан, то им становится созданный элемент. Функция возвращает созданный элемент. Возвращаемое значение функции – корень дерева.

if (root == NULL) return r; //если root == NULL, то до этого дерева не существовало, r становится корнем дерева

Определение расположения созданного узла происходит через сравнение числового значения созданного узла с числовым значением его родителя. Созданный узел становится родителем для следующего создаваемого узла. Родитель передается в параметрах при каждом вызове функции и содержится в *root*.

if (data > root->data) root->left = r; //левая ветка - значение больше текущего

else root->right = r; //правая ветка - значение меньше текущего

return r;

}

Случай *r != NULL* означает, что проход до конца дерева не окончен. Вызывается функция создания нового узла, где в первом параметре передается текущий узел (родитель для создаваемого узла), во втором – создаваемый, в третьем – числовое значение создаваемого узла.

//проход по всем элементам дерева, учитывая их вес

if (data > r->data) //левая ветка - значение больше текущего

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

else

CreateTree(r, r->right, data); //правая ветка - значение меньше текущего

Рекурсивная функция *print\_tree()* организует вывод бинарного дерева слева направо. Выход из функции происходит при нахождении нулевого узла

void print\_tree(struct Node\* r, int l){

if (r == NULL){ //конец дерева, выход из функции

return;

}

Сначала функция вызывается для вывода правой ветки дерева, затем – для левой. Параметр *l* передается в функции для организации вывода пробелом между числовыми значениями.

print\_tree(r->right, l + 1); //рекурсивный вызов функции для правой ветки

for (int i = 0; i < l; i++){ //цикл для организации вывода значений дерева

printf(" ");

}

printf("%d\n", r->data); //вывод значений дерева

print\_tree(r->left, l + 1); //рекурсивный вызов функции для левой ветки

}

Функция поиска *find()* принимает искомое значение, которое сравнивается с каждым значением узла дерева. При совпадении значений увеличивается счетчик *count*. Вызов функции также организован сначала для правой ветки дерева, затем – для левой. Выход из функции происходит при нахождении нулевого узла. Возвращаемое значение функции – число вхождений заданного элемента в дерево.

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

if (r == NULL){ //конец дерева, выход из функции

return count;

}

if (r->data == D\_find) count++; //элемент найден, увеличение счетчика count

find(r->right, D\_find); //рекурсивный вызов функции для правой ветки

find(r->left, D\_find); //рекурсивный вызов функции для левой ветки

}

В главной функции *main()* организован цикл по флагу *start*. Флаг сбрасывается в 0 при вводе пользователем значения «-1». Если пользователь вводит другое значения, вызывается функция создания нового узла дерева.

\* 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.

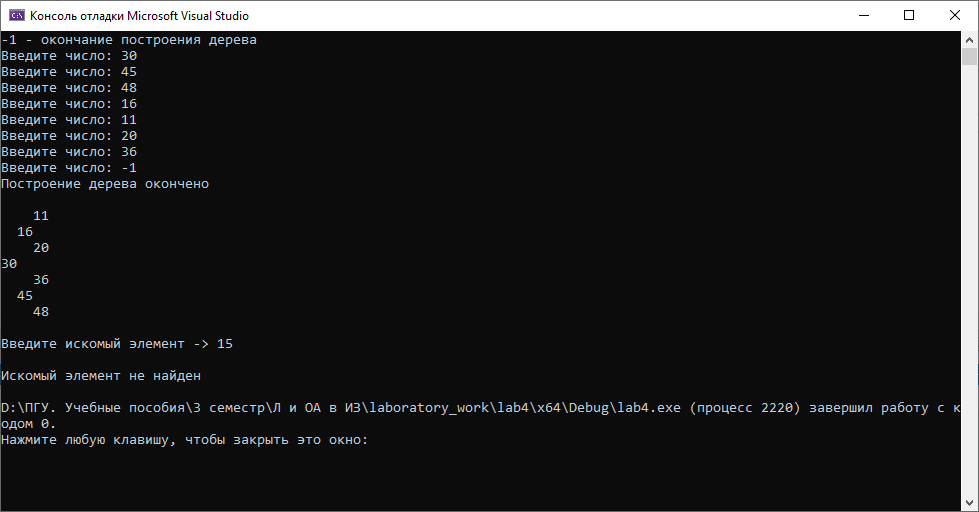


Рисунок 1 – поиск заданного элемента

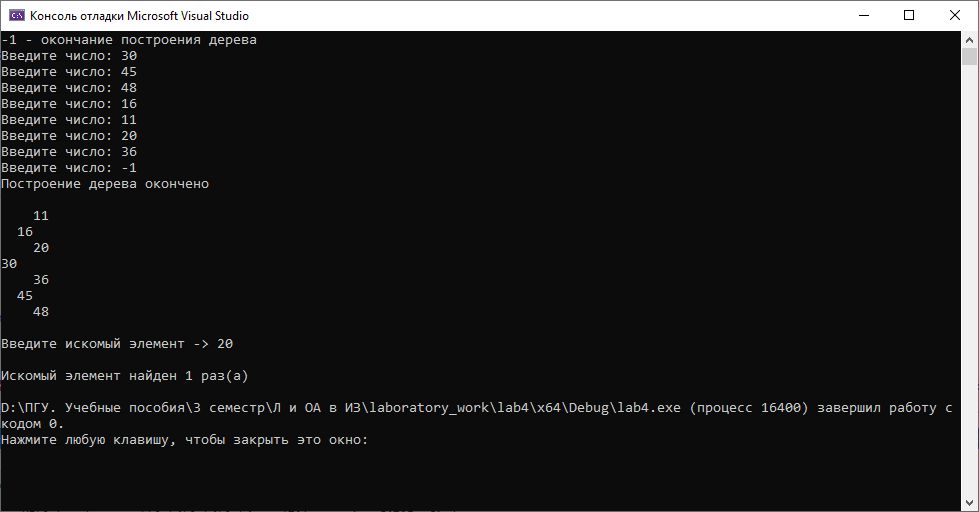


Рисунок 2 – поиск заданного элемента

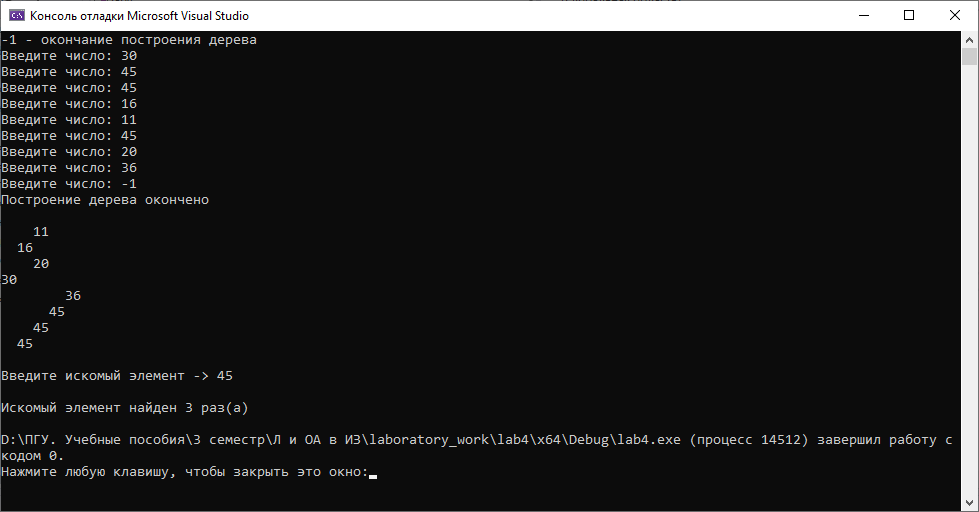


Рисунок 3 – поиск заданного элемента

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

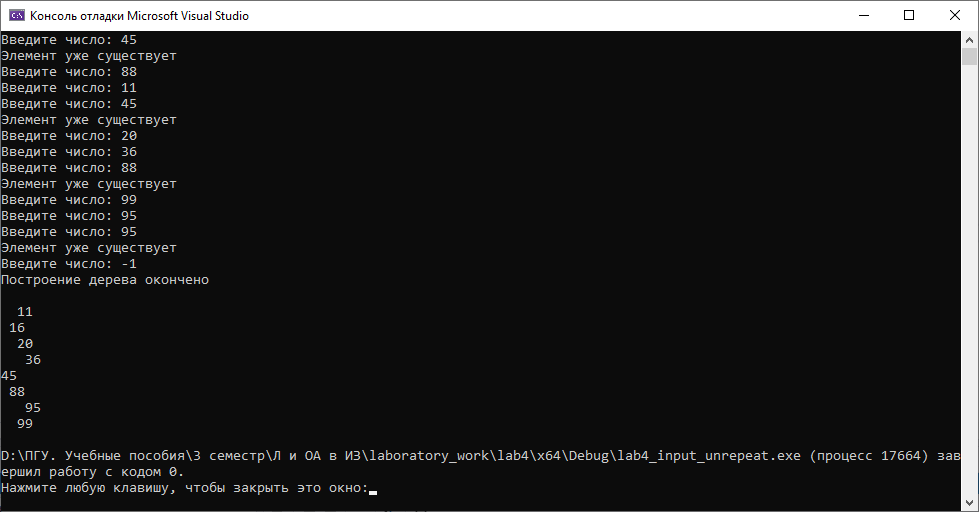


Рисунок 4 – добавление элементов без повторений

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;

}