Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Динамические списки»

Выполнили ст. группы 22ВВВ1:

Уткин М.М.

Саветкин Д.Д.

Соколовский Е.В

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. В.

Пенза 2023

**Цель работы:**

Цель данной лабораторной работы заключается в изучении и практическом применении бинарных деревьев поиска.

**Лабораторное задание:**

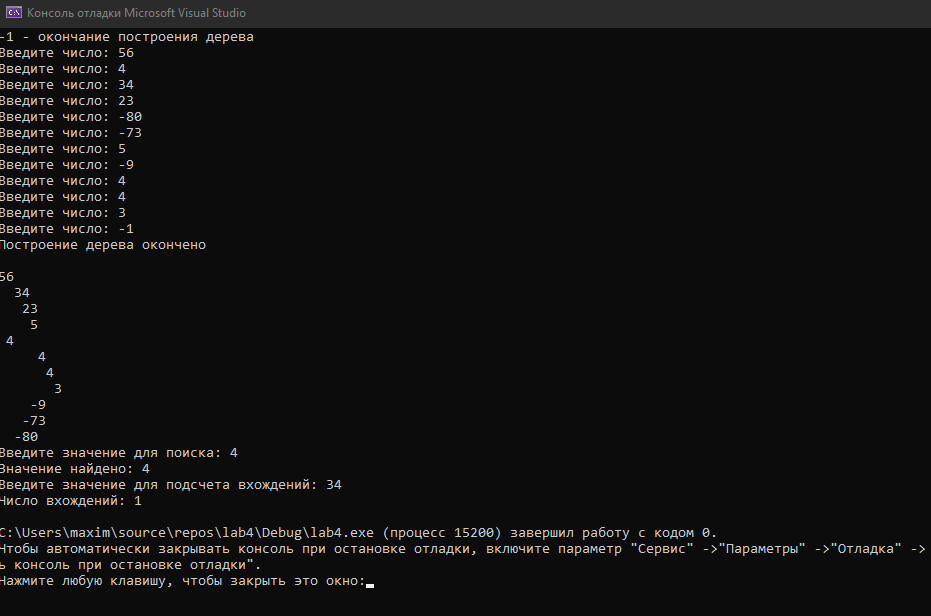
1. Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.
2. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.
3. \*Изменить функцию добавления элементов для исключения добавления одинаковых символов.
4. \*Оценить сложность процедуры поиска по значению в бинарном дереве.

**Ход работы:**

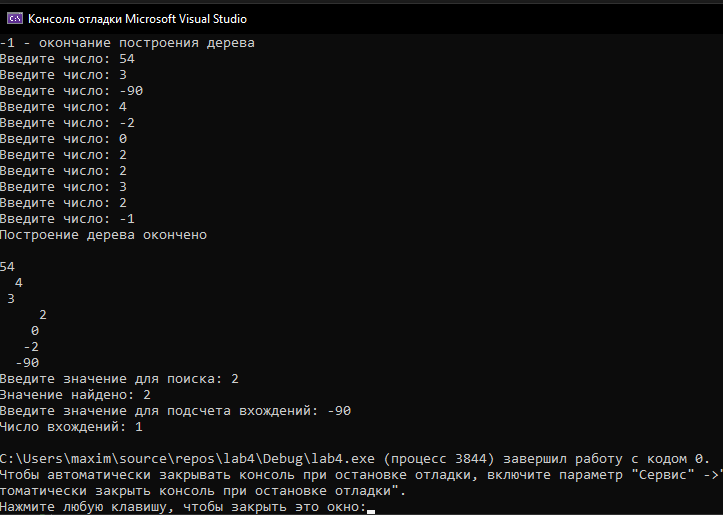
Объявляем структуру, которая представляет узел дерева. Создаем функции для построения, создания, вывода дерева на экран и поиска искомого значения. В главной функции вызываем заданные функции для заполнения, вывода дерева, поиска искомого значения и вывода количества вхождений определенного значения.

**Результаты работы программы:**

**Задания 1 - 2:**



**Задания 3:**



**Вывод:**

В ходе лабораторной работы нами было изучено бинарное дерево поиска, которое является структурой данных, используемой для хранения и поиска элементов. Реализовали такие функции как: построение, создание, вывод дерева на экран и поиск искомого значения. Оценили сложность кода, она равна O(log(n)).

**Листинг:**

**Код 1. Задания 1 – 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\* CreateTree(struct Node\* root, int data) {

if (root == NULL) {

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

if (newNode == NULL) {

printf("Ошибка выделения памяти");

exit(0);

}

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

if (data <= root->data) {

root->left = CreateTree(root->left, data);

}

else {

root->right = CreateTree(root->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);

}

struct Node\* Search(struct Node\* root, int value) {

if (root == NULL || root->data == value) {

return root;

}

if (value < root->data) {

return Search(root->left, value);

}

else {

return Search(root->right, value);

}

}

int CountOccurrences(struct Node\* root, int value) {

if (root == NULL) {

return 0;

}

int count = 0;

if (root->data == value) {

count = 1;

}

count += CountOccurrences(root->left, value);

count += CountOccurrences(root->right, value);

return count;

}

int main() {

setlocale(LC\_ALL, "RU");

int D, start = 1;

struct Node\* root = NULL; // Declare root node

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

while (start) {

printf("Введите число: ");

scanf("%d", &D);

if (D == -1) {

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

start = 0;

}

else {

root = CreateTree(root, D); // Update root with the new tree

}

}

print\_tree(root, 0);

printf("Введите значение для поиска: ");

scanf("%d", &D);

struct Node\* result = Search(root, D);

if (result != NULL) {

printf("Значение найдено: %d\n", result->data);

printf("Введите значение для подсчета вхождений: ");

int searchValue;

scanf("%d", &searchValue);

int count = CountOccurrences(root, searchValue);

printf("Число вхождений: %d\n", count);

}

else {

printf("Значение не найдено\n");

}

return 0;

}

**Код 2. Задания 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\* CreateTree(struct Node\* root, int data) {

if (root == NULL) {

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

if (newNode == NULL) {

printf("Ошибка выделения памяти");

exit(0);

}

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

if (data < root->data) {

root->left = CreateTree(root->left, data);

}

else if (data > root->data) {

root->right = CreateTree(root->right, data);

}

// Игнорируем добавление элемента с тем же значением,

// которое уже существует в дереве

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);

}

struct Node\* Search(struct Node\* root, int value) {

if (root == NULL || root->data == value) {

return root;

}

if (value < root->data) {

return Search(root->left, value);

}

else {

return Search(root->right, value);

}

}

int CountOccurrences(struct Node\* root, int value) {

if (root == NULL) {

return 0;

}

int count = 0;

if (root->data == value) {

count = 1;

}

count += CountOccurrences(root->left, value);

count += CountOccurrences(root->right, value);

return count;

}

int main() {

setlocale(LC\_ALL, "RU");

int D, start = 1;

struct Node\* root = NULL; // Declare root node

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

while (start) {

printf("Введите число: ");

scanf("%d", &D);

if (D == -1) {

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

start = 0;

}

else {

root = CreateTree(root, D); // Update root with the new tree

}

}

print\_tree(root, 0);

printf("Введите значение для поиска: ");

scanf("%d", &D);

struct Node\* result = Search(root, D);

if (result != NULL) {

printf("Значение найдено: %d\n", result->data);

printf("Введите значение для подсчета вхождений: ");

int searchValue;

scanf("%d", &searchValue);

int count = CountOccurrences(root, searchValue);

printf("Число вхождений: %d\n", count);

}

else {

printf("Значение не найдено\n");

}

return 0;

}