Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

Выполнили:

студенты группы 21ВВ2

Сорокина Е.А.

Нефедова Е.Д.

Принял:

Юрова О.В.

Пенза 2022

**Задание 1**:

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

**Задание 2**:

Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <cstdlib>

#include <locale.h>

struct Node {

Node\* left;

Node\* right;

int data;

};

struct Node\* root = NULL;

bool search(struct Node\* node, int value);

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;

}

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

{

if (!search(root, data)) { // если элемент не найден в дереве, то продолжаем как было

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)

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

else

CreateTreeUnique(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);

}

// поиск

bool search(struct Node\* node, int value) {

if (node == NULL) {

return false;

}

if (node->data == value)

return true; // нашли

if (value < node->data) // если искомое меньше значения в текущем узле то ищем в правом поддереве перевернутого дерева

return search(node->right, value);

if (value > node->data) // если искомое больше значения в текущем узле то ищем в левом поддереве перевернутого дерева

return search(node->left, value);

}

// подсчет числа повторений

int count\_repeats(struct Node\* node, int val) {

if (node == NULL)

return 0;

if (node->data == val) { // нашли искомый, считаем его повторения в правых узлах начиная с текущего

struct Node\* current = node;

int counter = 1;

while ((current = current->right) != NULL && current->data == val)

counter++;

return counter;

}

else if (val > node->data) {

return count\_repeats(node->left, val); // ищем в левом поддереве

}

else if (val < node->data) {

return count\_repeats(node->right, val); // ищем в правом поддереве

}

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1;

root = NULL;

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, root, D);

// root = CreateTreeUnique(root, root, D);

}

}

print\_tree(root, 0);

// Поиск

printf("Введите число для поиска: ");

scanf("%d", &D);

if (D != -1) {

if (search(root, D)) {

printf("Найдено\n");

printf("Число повторений: %d\n", count\_repeats(root, D));

}

else

printf("Не найдено\n");

}

return 0;

}

3. \* Изменить функцию добавления элементов для исключения добавления одинаковых символов.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <cstdlib>

#include <locale.h>

struct Node {

Node\* left;

Node\* right;

int data;

};

struct Node\* root = NULL;

bool search(struct Node\* node, int value);

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;

}

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

{

if (!search(root, data)) { // если элемент не найден в дереве, то продолжаем как было

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)

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

else

CreateTreeUnique(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);

}

// поиск

bool search(struct Node\* node, int value) {

if (node == NULL) {

return false;

}

if (node->data == value)

return true; // нашли

if (value < node->data) // если искомое меньше значения в текущем узле то ищем в правом поддереве перевернутого дерева

return search(node->right, value);

if (value > node->data) // если искомое больше значения в текущем узле то ищем в левом поддереве перевернутого дерева

return search(node->left, value);

}

// подсчет числа повторений

int count\_repeats(struct Node\* node, int val) {

if (node == NULL)

return 0;

if (node->data == val) { // нашли искомый, считаем его повторения в правых узлах начиная с текущего

struct Node\* current = node;

int counter = 1;

while ((current = current->right) != NULL && current->data == val)

counter++;

return counter;

}

else if (val > node->data) {

return count\_repeats(node->left, val); // ищем в левом поддереве

}

else if (val < node->data) {

return count\_repeats(node->right, val); // ищем в правом поддереве

}

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1;

root = NULL;

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, root, D);

root = CreateTreeUnique(root, root, D);

}

}

print\_tree(root, 0);

// Поиск

printf("Введите число для поиска: ");

scanf("%d", &D);

if (D != -1) {

if (search(root, D)) {

printf("Найдено\n");

printf("Число повторений: %d\n", count\_repeats(root, D));

}

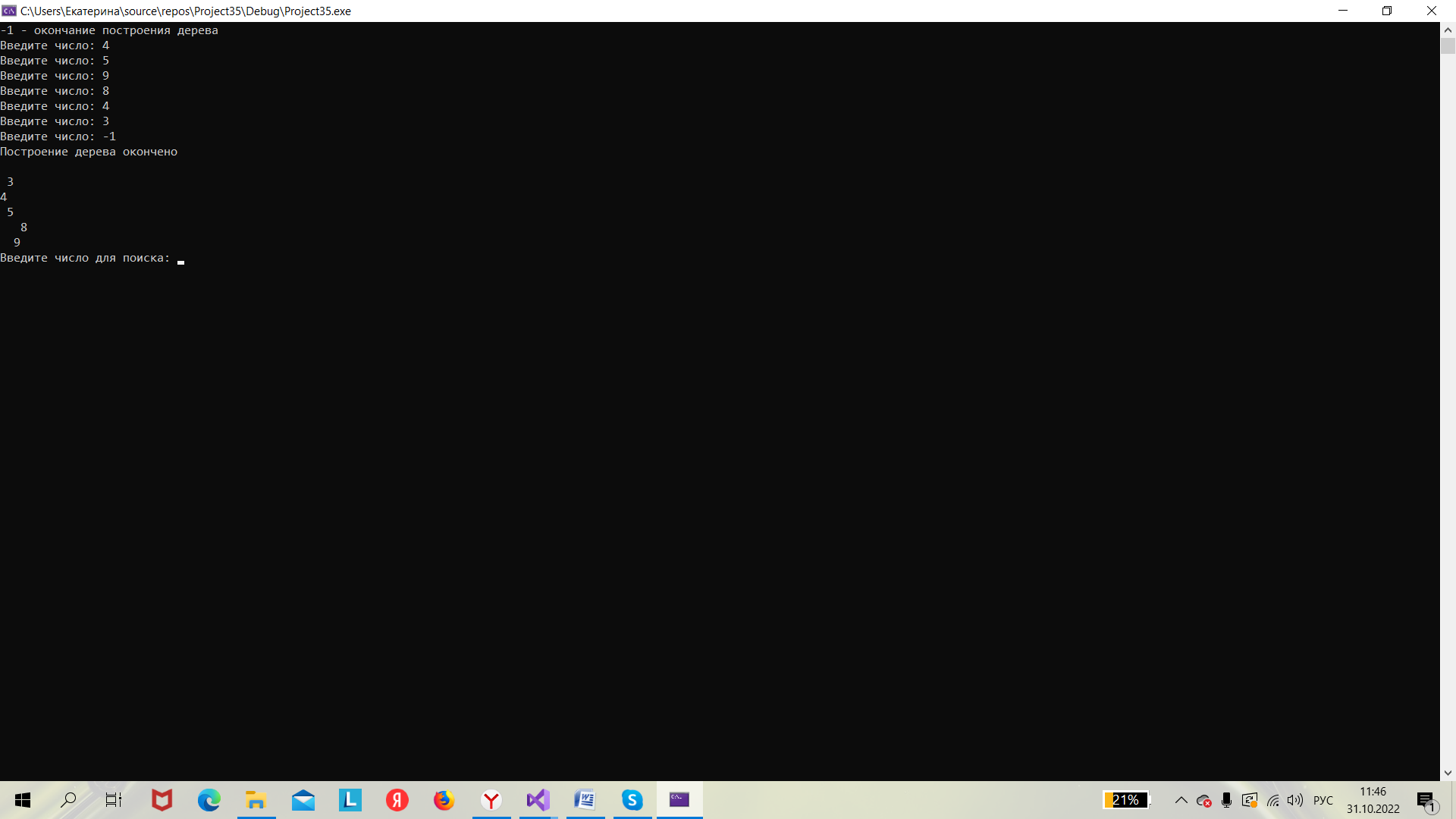
else

printf("Не найдено\n");

}

return 0;

}



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

Сложность поиска по значению в бинарном дереве определяется по формуле .

Для данной программы сложность равна

**Вывод:** в ходе данной лабораторной работы мы научились реализовывать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве и вести подсчёт повторяющихся элементов, а также исключать повторяющееся значение, при выводе исключая его.