Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах.»

на тему: «Бинарное дерево поиска.»

Выполнили**:**

студенты группы 21ВВ4

Федоренко Вероника

Роганов Данила

Проверили:

Юрова О.В,

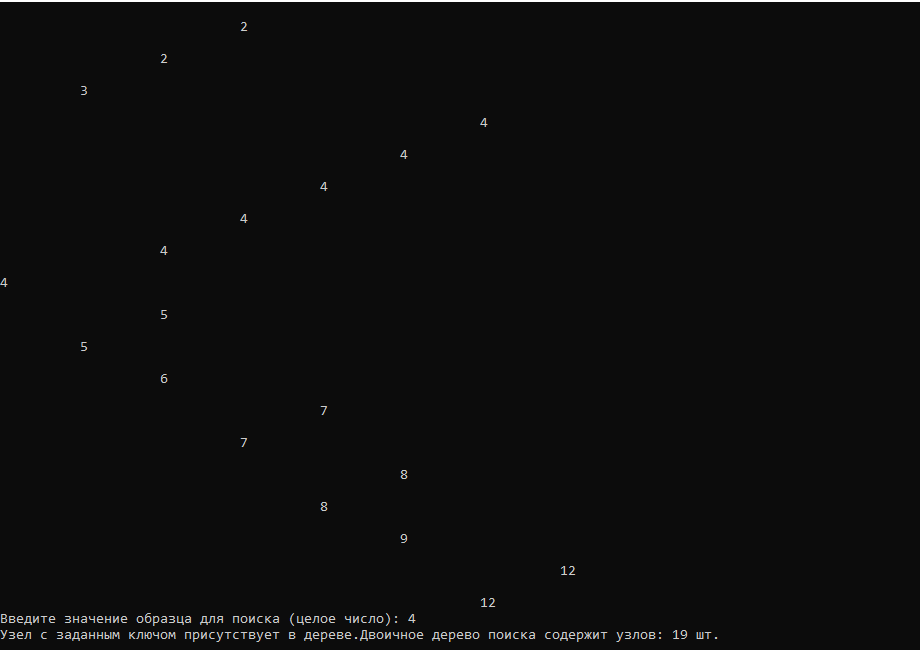
Акифьев И.В.

Пенза 2022

**Лабораторное задание**

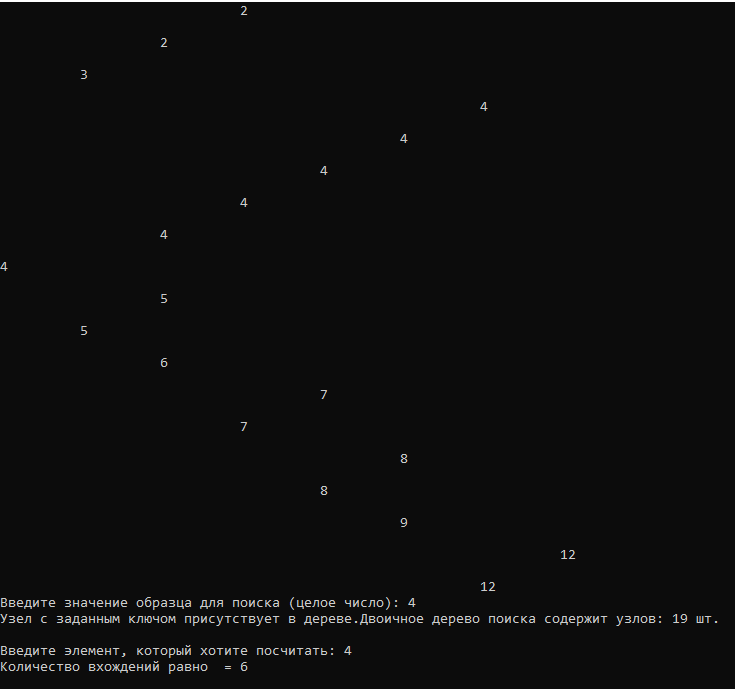
Задание 1. Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.

**Результат**



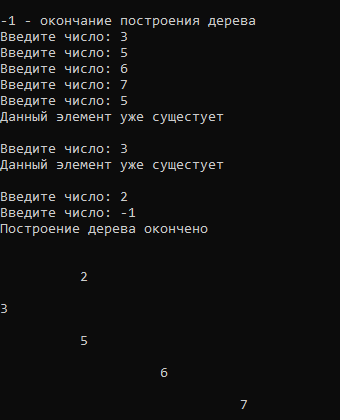
Задание 2. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

**Результат**



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

**Результат**



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

**Результат**

Алгоритм поиска в дереве имеет логарифмическую сложность O(log n).

**Листинг**

#include <iostream>

#include <conio.h>

#include <stdlib.h>

#include <stdio.h>

#include <clocale>

//#include <malloc.h>

using namespace std;

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root;

Node\* CreateTree(struct Node\* root, struct Node\* userElement, int data)

{

if (userElement == NULL)

{

userElement = (Node\*)malloc(sizeof(Node));

if (userElement == NULL)

{

printf("Ошибка выделения памяти");

exit(0);

}

userElement->left = NULL;

userElement->right = NULL;

userElement->data = data;

if (root == NULL) {

return userElement;

}

if (data > root->data) {

root->left = userElement;

}

else {

root->right = userElement;

}

return userElement;

}

if (data > userElement->data) {

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

}

else {

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

}

return root;

}

//форматирование дерева

void print2DUtil(struct Node\* root, int space)

{

const int scpaces = 10;

// Base case

if (root == NULL)

return;

// Increase distance between levels

space += scpaces;

// Process right child first

print2DUtil(root->right, space);

// Print current node after space

// count

printf("\n");

for (int i = scpaces; i < space; i++)

printf(" ");

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

// Process left child

print2DUtil(root->left, space);

}

// поиска образца в дереве по заданному ключу узла

bool SearchNodeByKey(Node\* root, const int key)

{

if (root == NULL)

return false;

if (root->data == key)

return true;

if (key > root->data) {

return (SearchNodeByKey(root->left, key));

}

else {

return (SearchNodeByKey(root->right, key));

}

}

//поиск количества вхождений элемента

int foundCountSameElementsInBT(Node\* root, const int key)

{

if (root == NULL) {

return 0;

}

int incremeter = 0;

if (root->data == key) {

incremeter = 1;

}

return incremeter + foundCountSameElementsInBT(root->right, key) +

foundCountSameElementsInBT(root->left, key);

}

// подсчет количество узлов дерева (рекурсивный обход)

int GetCountNodes(Node\* root)

{

if (root == NULL)

return 0;

else

return GetCountNodes(root->left) + GetCountNodes(root->right) + 1;

}

//добавление элементов в дерево без повторений

Node\* AddElementToTreeWithOutRepeat(struct Node\* localRoot, struct Node\* userElement, int data)

{

if (SearchNodeByKey(localRoot, data)) {

return NULL;

}

if (userElement == NULL)

{

userElement = (Node\*)malloc(sizeof(Node));

if (userElement == NULL)

{

printf("Ошибка выделения памяти");

exit(0);

}

userElement->left = NULL;

userElement->right = NULL;

userElement->data = data;

if (localRoot == NULL) {

return userElement;

}

if (data > localRoot->data) {

localRoot->left = userElement;

}

else {

localRoot->right = userElement;

}

return userElement;

}

if (data > userElement->data) {

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

}

else {

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

}

return localRoot;

}

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);

}

print2DUtil(root, 0);

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

//--------------------------------------------

int userInput;

if (root == NULL)

printf("Бинарное дерево поиска не содержит ни одного узла. Поиск образца невозможен. \n");

else

{

printf("Введите значение образца для поиска (целое число): ");

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

if (SearchNodeByKey(root, userInput))

{

printf("Узел с заданным ключом присутствует в дереве.");

}

else

{

printf("Узел с заданным ключом отсутсвует в дереве.");

}

}

//-----------------------------------------------

printf("Двоичное дерево поиска содержит узлов: %d шт.\n", GetCountNodes(root));

if (root == NULL) {

printf("\nБинарное дерево поиска не содержит ни одного узла. Поиск образца невозможен. \n");

}

else

{

printf("\nВведите элемент, который хотите посчитать: ");

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

printf("Количество вхождений равно = %d", foundCountSameElementsInBT(root, userInput));

}

//создание нового дерева без повторений

printf("\n\n\n\n\n\n\n\n\n\n\n\n");

Node\* rootTree = NULL;

start = 1;

printf("\n-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 {

Node\* elementToAdd = AddElementToTreeWithOutRepeat(rootTree, rootTree, D);

if (elementToAdd == NULL) {

printf("Данный элемент уже сущестует\n\n");

}

else {

rootTree = elementToAdd;

}

}

}

print2DUtil(rootTree, 0);

return 0;

}

**Вывод**: в ходе выполнения лабораторной работы были функции для работы с бинарными деревьями.