МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнил:

Студент группы 23ВВВ2

Монин Иван

Приняли:

доцент Юрова О.В.

доцент Митрохин М.А.

Пенза 2024

**Цель работы** – научиться создавать бинарные деревья и реализовать стандартные функции для них

**Лабораторное задание:**

**Задание 1**

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

**Задание 2**

Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

**Задание 3**\*

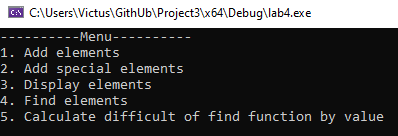
Изменить функцию добавления элементов для исключения добавления одинаковых символов.

**Задание 4**\*

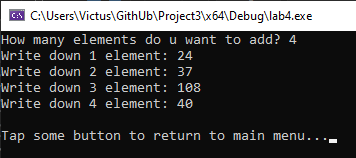
Оценить сложность процедуры поиска по значению в бинарном дереве.

**Задание 1**

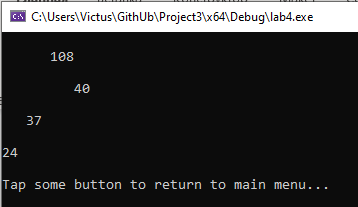
1. В начале предлагается меню с выбором:



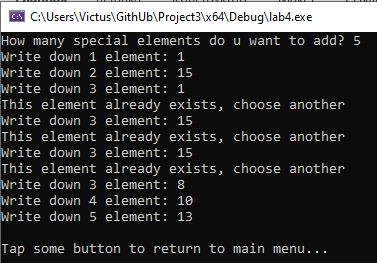
1. При выборе добавления элементов нужно указать кол-во добавляемых элементов, а затем ввести значения



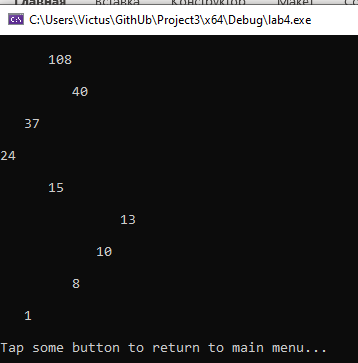
1. При выборе отображения элементов откроется дерево всех добавленных значений



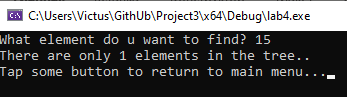
1. При выборе 2 пункта можно вводить уникальные значения. Если вводимый элемент уже есть в дереве, то будет выводиться соответствующее сообщение



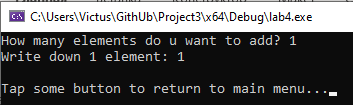
Посмотрим, как преобразилось дерево после добавления новых элементов (кнопка 3):



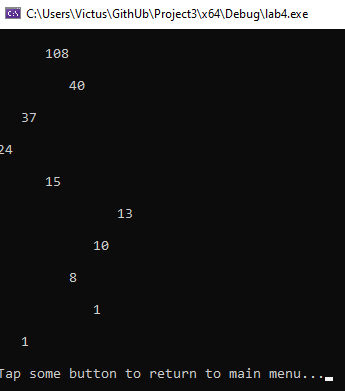
1. При поиске элемента необходимо ввести значение



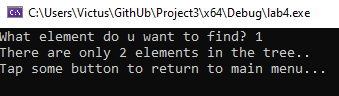
Выводится количество элементов с таким значением. Добавим ещё раз элемент, который уже есть в дереве(кнопка 1):



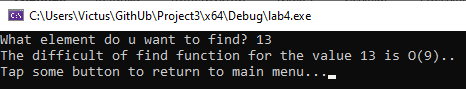
Воспользуемся отображением дерева(кнопка 3):

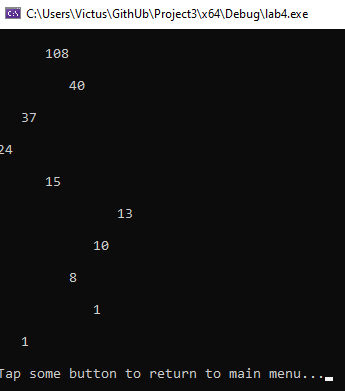


Видим, что элемент добавлен, теперь попробуем посчитать кол-во “1” с помощью функции(кнопка 4):



1. Также можно посчитать сложность функции поиска по определенному значению





О(8)

О(7)

О(6)

О(5)

О(4)

О(3)

О(2)

О(1)

О(9)

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

typedef struct Tree {

int data;

struct Tree\* left;

struct Tree\* right;

} Tree;

Tree\* addElem(Tree\* root, int data) {

Tree\* tmp = NULL;

if (!(tmp = (Tree\*)malloc(sizeof(sizeof(Tree))))) {

printf("Memory error");

exit(1);

}

tmp->data = data;

tmp->left = NULL;

tmp->right = NULL;

if (root == NULL) {

root = tmp;

return root;

}

if (data < root->data) {

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

}

else {

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

}

return root;

}

void printTree(Tree\* root,int size) {

if (root == NULL) {

return;

}

printTree(root->right, size + 1);

printf("\n");

for (int i = 0; i < size; i++) {

printf(" ");

}

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

printTree(root->left, size + 1);

}

int find(Tree\* root, int data,int count) {

if (root == NULL) {

return count;

}

if (root->data == data)count++;

count =find(root->right, data,count);

count =find(root->left, data, count);

return count;

}

int calcDiff(Tree\* root, int data, int count) {

if (root == NULL) {

return count;

}

count++;

if (root->data == data) printf("The difficult of find function for the value %d is O(%d)..\n", data, count);

count = calcDiff(root->right, data, count);

count = calcDiff(root->left, data, count);

return count;

}

Tree\* addSpecialElem(Tree\* root, int data) {

if (find(root, data, 0) != 0) {

printf("This element already exists, choose another\n");

return root;

}

Tree\* tmp = NULL;

if (!(tmp = (Tree\*)malloc(sizeof(sizeof(Tree))))) {

printf("Memory error");

exit(1);

}

tmp->data = data;

tmp->left = NULL;

tmp->right = NULL;

if (root == NULL) {

root = tmp;

return root;

}

if (data < root->data) {

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

}

else {

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

}

return root;

}

void displayMenu(Tree\* root) {

int data = 0;

int count;

printf("----------Menu----------\n");

printf("1. Add elements\n");

printf("2. Add special elements\n");

printf("3. Display elements\n");

printf("4. Find elements\n");

printf("5. Calculate difficult of find function by value\n");

int choice;

scanf("%d", &choice);

system("cls");

switch (choice) {

case(1):

printf("How many elements do u want to add? ");

scanf("%d", &count);

for (int i = 0; i < count; i++) {

printf("Write down %d element: ", i + 1);

scanf("%d", &data);

if (root == NULL)

root = addElem(root, data);

else addElem(root, data);

}

printf("\nTap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(root);

break;

case(2):

printf("How many special elements do u want to add? ");

scanf("%d", &count);

for (int i = 0; i < count; i++) {

printf("Write down %d element: ", i + 1);

scanf("%d", &data);

if (find(root, data, 0) != 0) i--;

if (root == NULL)

root = addSpecialElem(root, data);

else addSpecialElem(root, data);

}

printf("\nTap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(root);

break;

case(3):

printTree(root,0);

printf("\nTap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(root);

break;

case(4):

printf("What element do u want to find? ");

scanf("%d", &data);

count = find(root, data, 0);

printf("There are only %d elements in the tree..\n", count);

printf("Tap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(root);

break;

case(5):

printf("What element do u want to find? ");

scanf("%d", &data);

count = calcDiff(root, data, 0);

printf("Tap some button to return to main menu...");

getchar();

getchar();

system("cls");

displayMenu(root);

break;

default:

printf("Wrong button!");

displayMenu(root);

break;

}

}

void main() {

Tree\* root = NULL;

displayMenu(root);

}

**Вывод**

В ходе работы выполнения лабораторной работы были реализованы стандартный функции дерева. Так же была добавлена функция измерения сложности алгоритма.