Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу “Логика и основа алгоритмизации в ИЗ”

на тему “Бинарное дерево поиска”

Выполнили студенты группы 21ВВ1.3:

Лукьянова Дарья

Давкин Максим

Михальцова Полина

Приняли:

Митрохин М.А.

Юрова О.В.

Пенза 2022

**Название:**

Бинарное дерево 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.

**Листинг:**

**Программа к заданиям 1,2:**

#include <stdio.h>

#include <conio.h>

#include <malloc.h>

#include <locale.h>

#include <windows.h>

#pragma warning (disable:6054)

struct tree {

int data;

struct tree\* right;

struct tree\* left;

};

struct tree\* root = NULL;

// функция создания дерева

struct tree\* CreateTree(struct tree\* root, struct tree\* r, int data)

{

if (r == NULL)

{

r = (struct tree\*)malloc(sizeof(struct tree));

if (r == NULL)

{

printf("Ошибка выделения памяти");

exit(1);

}

r->right = NULL;

r->left = NULL;

r->data = data;

if (root == NULL)

return r;

if (data > root->data)

root->right = r;

else root->left = r;

return r;

}

if (data > r->data)

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

else

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

return root;

}

// функция вывода дерева на экран

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

{

if (r == NULL)

{

return;

}

print\_tree(r->left, l + 1);

for (int i = 0; i < l; i++)

{

printf(" ");

}

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

print\_tree(r->right, l + 1);

}

// функция поиска заданного элемента в дереве

int search(tree\* root, int key)

{

if (root == NULL) // если дерево пустое или закончен обход

return 0;

if (root->data == key) // если найден искомый элемент

return root->data;

if (key < root->data) // поиск по дереву

return search(root->left, key);

else return search(root->right, key);

return 0;

}

// функция подсчета числа вхождений элемента в дерево

void funcchet(struct tree\*\* root, int chet, int m) {

if (\*root != NULL) {

if (chet == (\*root)->data)

m++;

if (chet > (\*root)->data)

funcchet(&((\*root)->right), chet, m);

else funcchet(&((\*root)->left), chet, m);

return;

}

printf("Число вхождений данного элемента в дерево: %d\n\n", m);

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1, key, chet, m, k;

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("\n1 - найти заданный элемент в дереве\n2 - посчитать число вхождений элемента в дерево\n3 - выход\n\n");

while (true) {

int l = \_getch();

switch (l) {

case '1':

printf("Введите элемент поиска: ");

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

k = search(root, key);

if (k == key)

printf("Данный элемент найден\n\n");

else printf("Данный элемент не найден\n\n");

break;

case '2':

printf("Введите элемент для подсчета числа его вхождений в дерево: ");

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

m = 0;

funcchet(&root, chet, m);

break;

case '3':

exit(1);

break;

default:

break;

}

}

return 0;

}

**Программа к заданию 3:**

#include <stdio.h>

#include <conio.h>

#include <malloc.h>

#include <locale.h>

#include <windows.h>

#pragma warning (disable:6054)

struct tree {

int data;

struct tree\* right;

struct tree\* left;

};

struct tree\* root = NULL;

// функция создания дерева

struct tree\* CreateTree(struct tree\* root, struct tree\* r, int data)

{

if (r == NULL)

{

r = (struct tree\*)malloc(sizeof(struct tree));

if (r == NULL)

{

printf("Ошибка выделения памяти");

exit(1);

}

r->right = NULL;

r->left = NULL;

r->data = data;

if (root == NULL)

return r;

if (data > root->data)

root->right = r;

else root->left = r;

return r;

}

if (data > r->data)

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

else if (data < r->data) CreateTree(r, r->left, data);

else return root;

return root;

}

// функция вывода дерева на экран

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

{

if (r == NULL)

{

return;

}

print\_tree(r->left, l + 1);

for (int i = 0; i < l; i++)

{

printf(" ");

}

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

print\_tree(r->right, l + 1);

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1;

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;

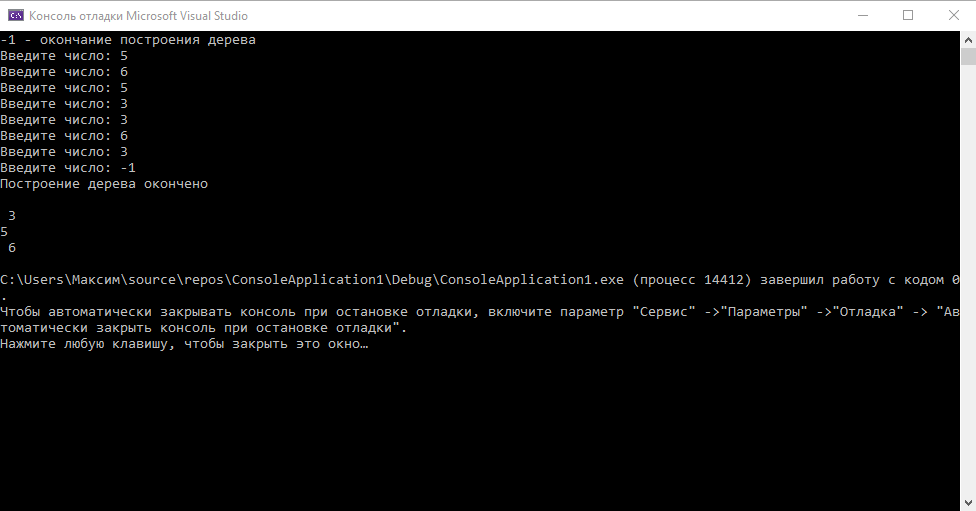
}

**Задание 4:**

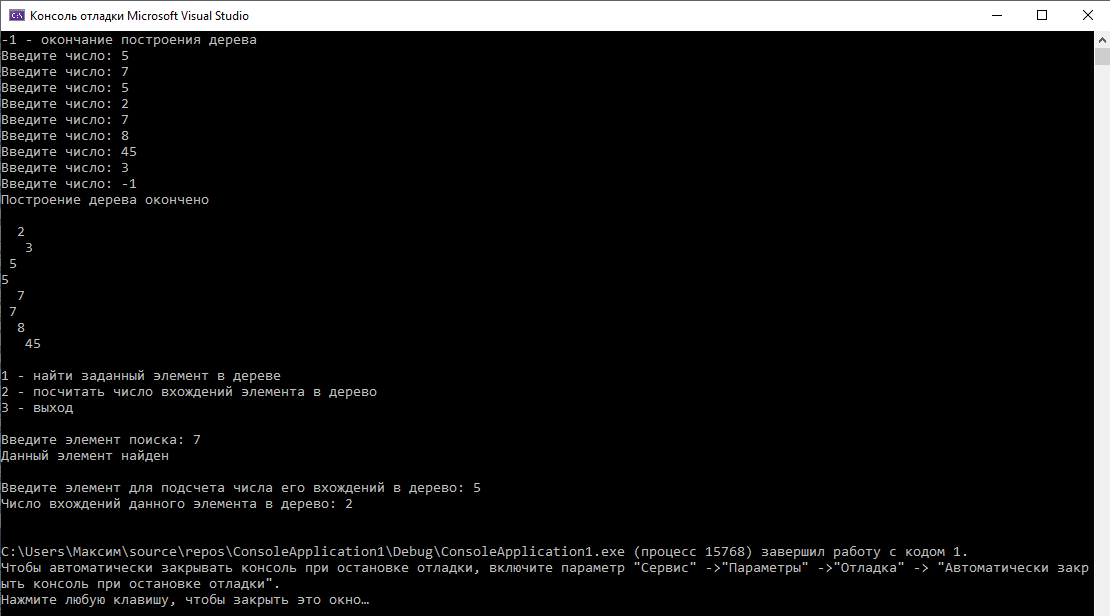
В бинарном поиске значение введенного пользователем элемента сравнивается со значением корня: если корень оказался больше, то введенный элемент далее будет сравниваться со значением левого потомка, иначе – правого, тем самым в обоих случаях отсекается область поиска в два раза (в оптимальном случае, когда глубина всех листьев одинакова). Далее рекурсия – значение каждого следующего узла будет сравниваться с введенным пользователем элементом до тех пор, пока не будет найдено соответствие. Т.е. количество непроверенных значений на i-ом рекурсивном вызове: n / 2i. В какой-то момент данное выражение достигнет единицы: 2i = n, из чего справедливо: i = log n. Сложность алгоритма поиска по значению в бинарном дереве – 𝑂 (log 𝑛).

**Результаты выполнения программ:**

**Рисунок 2 – результат выполнения программы к третьему заданию**



**Рисунок 1 – результат выполнения программы к первому и второму заданию**



**Вывод:** в ходе выполнения лабораторной работы были реализованы алгоритм поиска элемента по значению в дереве, функция подсчета числа вхождений заданного элемента в дерево; оценена сложность алгоритма поиска по значению в дереве, была изменена программа таким образом, что возможность добавления одинаковых элементов в дерево исключена.