Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнили:

студенты группы 21ВВ1.2

Сагателов А.К.  
Митрошин А.Д.

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.

**Листинг**

Задание 1 и 2

#include "stdafx.h"

#include <string.h>

#include <memory.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

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->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;

}

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);

}

void search\_elem(struct Node\* r, int data) //поиск по заданному элементу

{

if (r == NULL)

return;

if (r->data < data) {

search\_elem(r->right, data);

}

else if (r->data > data) {

search\_elem(r->left, data);

}

if (r->data == data)

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

}

int counter(Node\* root, int E) //счётчик

{

if (root == 0) return 0;

return (root->data == E) + counter(root->left, E) + counter(root->right, E);

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int D, number;

struct Node\* root = NULL;

printf("Введите размер дерева: ");

scanf("%d", &D);

while (D) {

root = CreateTree(root, root, rand() % 100);

D--;

}

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

print\_tree(root, 0);

printf("\nВведите элемент для поиска и подсчёта: ");

scanf("%d", &D);

search\_elem(root, D);

number = counter(root, D);

if (number)

printf("Кол-во искомых элементов: %d\n", number);

else

printf("Ничего не найдено\n");

system("pause");

return 0;

}

Задание 3

#include "stdafx.h"

#include <string.h>

#include <memory.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

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->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;

}

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);

}

void search\_elem(struct Node\* r, int data) //поиск по заданному элементу

{

if (r == NULL)

return;

if (r->data < data) {

search\_elem(r->right, data);

}

else if (r->data > data) {

search\_elem(r->left, data);

}

if (r->data == data)

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

}

int checker(Node\* root, int E)

{

if (root == 0) return 0;

return (root->data == E) + checker(root->left, E) + checker(root->right, E);

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int D, number;

struct Node\* root = NULL;

printf("Введите размер дерева: ");

scanf("%d", &D);

while (D) {

do {

number = rand() % 100;

} while(checker(root, number));

root = CreateTree(root, root, number);

D--;

}

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

print\_tree(root, 0);

printf("\nВведите элемент для поиска: ");

scanf("%d", &D);

search\_elem(root, D);

system("pause");

return 0;

}

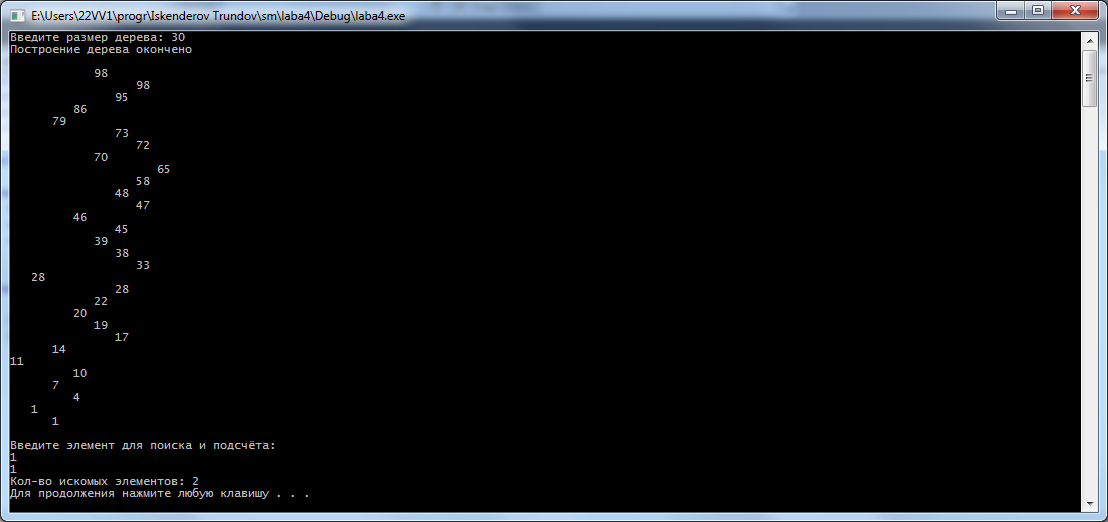
Задание 4

Временная сложность с линейной структурой — O(n), где n увеличивается по мере увеличения числа элементов для поиска.

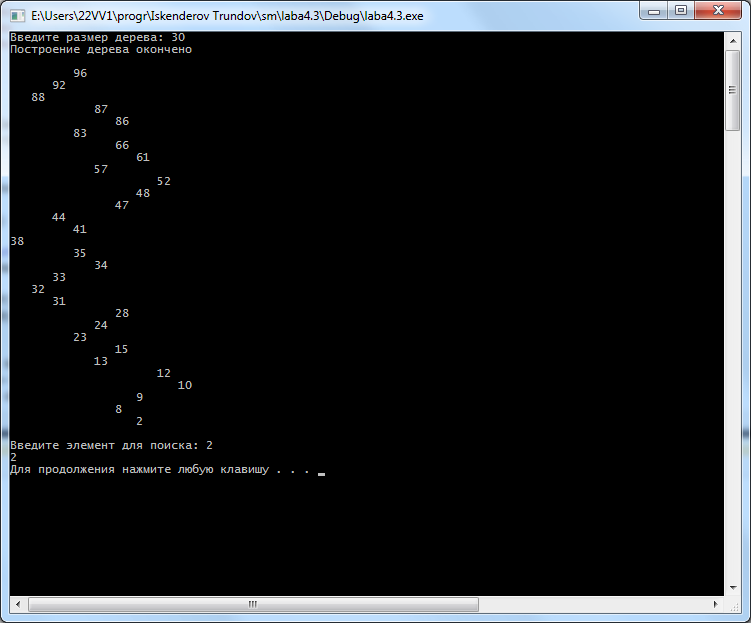
Данные представлены в виде древовидной структуры. Тогда число шагов, необходимых для нахождения значения, уменьшается более чем наполовину: логарифмическая временная сложность O(n\*log(n). Это происходит потому, что всегда, когда значение вставляется в двоичное дерево поиска, добавление узла упорядоченно: значение левого дочернего элемента меньше значения родительского, а значение правого узла больше значения родительского узла. Когда мы ищем значение, если оно меньше корня, мы игнорируем всю правую часть дерева и рекурсивно повторяем поиск в левой части дерева.

**Результат работы программы**

Задание 1 и 2



Задание 3

****

### Выводы

В ходе данной работы мы научились реализовывать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве, функцию подсчёта числа вхождений заданного элемента в дерево, изменить функцию добавления элементов для исключения добавления.