МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №4

по курсу «Л и ОА в ИЗ»

на тему «Бинарное дерево поиска»

Выполнили:

Студенты группы 22ВВС1

Костин Максим

Макеева Дарья

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза 2023

**Лабораторноезадание*:***

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:**

**С повторяющимися числами:**

#include <stdlib.h>

#include <conio.h>

#include <stdio.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

int n = 0;

struct Node\* root;

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 preOrderTravers(Node\* root, int d) {

if (root) {

if (d == root->data)

n++;

preOrderTravers(root->left, d);

preOrderTravers(root->right, d);

}

}

int main() {

setlocale(LC\_ALL, "Rus");

int D, start = 1, k;

int serc = 0;

root = NULL;

while (1) {

printf("\nВведите номер операции:\n1)Добавить эл. в дерево 2)Поиск эл. в дереве 3)Просмотреть эл. дерева 4)Посмотреть кол-во входов эл. 5)Выйти\n");

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

switch (k) {

case 1:

start = 1;

printf("-1 - окончание построения дерева\n");

while (start)

{

printf("Введите число: ");

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

if (D == -1)

{

start = 0;

}

else

root = CreateTree(root, root, D);

}

break;

case 2:

if (root != NULL) {

printf("Введите номер нужного элемента\n");

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

preOrderTravers(root, serc);

if (n > 0) {

printf("Элемент есть в дереве");

n = 0;

break;

}

else {

printf("Элемента нет в дереве");

break;

}

break;

}

else

printf("Нет элементов в дереве");

break;

case 3:

if (root != NULL) {

print\_tree(root, 3);

break;

}

else

printf("Нет элементов в дереве");

break;

case 4:

if (root != NULL) {

printf("Введите номер нужного элемента\n");

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

preOrderTravers(root, serc);

printf("Кол-во вхождений элемента - %d", n);

n = 0;

break;

}

else

printf("Нет элементов в дереве");

break;

case 5:

return 0;

}

}

}

**Без повторения чисел:**

#include <stdlib.h>

#include <conio.h>

#include <stdio.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

int n = 0;

struct Node\* root;

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 preOrderTravers(Node\* root,int d) {

if (root) {

if (d == root->data)

n++;

preOrderTravers(root->left, d);

preOrderTravers(root->right, d);

}

}

int main(){

setlocale(LC\_ALL, "Rus");

int D, start = 1, k;

int serc = 0;

root = NULL;

while (1) {

printf("\nВведите номер операции:\n1)Добавить эл. в дерево 2)Поиск эл. в дереве 3)Просмотреть эл. дерева 4)Посмотреть кол-во входов эл. 5)Выйти\n");

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

switch (k) {

case 1:

start = 1;

printf("-1 - окончание построения дерева\n");

while (start)

{

printf("Введите число: ");

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

if (D == -1)

{

start = 0;

}

else {

preOrderTravers(root, D);

if(n == 0)

root = CreateTree(root, root, D);

else {

printf("Ошибка! Введите неповторяющееся число: ");

n = 0;

break;

}

}

}

break;

case 2:

if (root != NULL) {

printf("Введите номер нужного элемента\n");

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

preOrderTravers(root, serc);

if (n > 0) {

printf("Элемент есть в дереве");

n = 0;

break;

}

else {

printf("Элемента нет в дереве");

break;

}

break;

}

else

printf("Нет элементов в дереве");

break;

case 3:

if (root != NULL) {

print\_tree(root, 3);

break;

}

else

printf("Нет элементов в дереве");

break;

case 4:

if (root != NULL) {

printf("Введите номер нужного элемента\n");

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

preOrderTravers(root, serc);

printf("Кол-во вхождений элемента - %d",n);

n = 0;

break;

}

else

printf("Нет элементов в дереве");

break;

case 5:

return 0;

}

}

}

**Оценка сложности процедуры поиска по значению в бинарном дереве:**

Временная сложность процедуры поиска в двоичном дереве зависит от формы

дерева. В худшем случае, когда двоичное дерево искажено и напоминает

связанный список, временная сложность процедуры поиска будет равна O(n),

где n - количество узлов в дереве. Это связано с тем, что в этом случае нам,

возможно, придется посетить каждый узел в дереве, прежде чем найти целевое

значение.

Однако в среднем и наилучшем случае, когда двоичное дерево

сбалансировано, временная сложность процедуры поиска будет равна O(log

n), где n - количество узлов в дереве. Это связано с тем, что на каждом шаге

поиска мы можем отбросить половину оставшихся узлов в дереве.

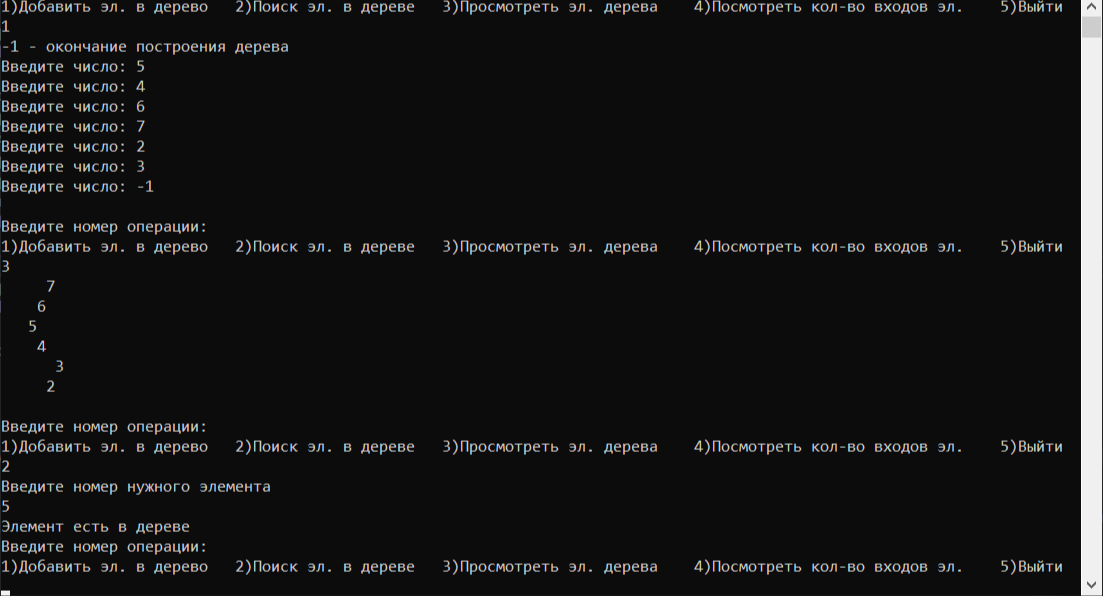
Таким образом, общая сложность процедуры поиска в двоичном дереве может

быть описана как O(log n) в среднем и наилучшем случаях и O(n) в наихудшем.

# Пояснительный текст к программе:

preOrderTravers(root, D) – функция для рекурсивного обхода дерева в глубину начиная с корня потом левое поддерево и правое.

**Результат работы программы**

****

**Вывод**

В ходе выполнения лабораторной работы научились работать с бинарным деревом поиска, реализовали поиск числа в дереве и количества вхождений указанного элемента. Также оценили сложность процедуры поиска по значению.