Министерство науки и высшего образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Бинарное дерево поиска»

Выполнили ст. группы 22ВВВ1:

Лёвин А.Д.

Колобов И.О.

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. В.

Пенза 2023

**Цель работы:**

Цель данной лабораторной работы заключается в изучении и практическом применении бинарных деревьев в программировании .

**Лабораторное задание:**

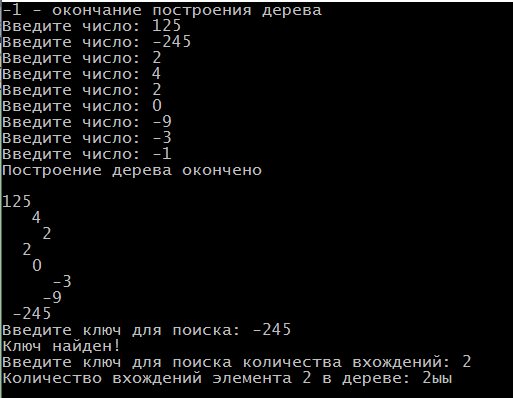
1. Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.
2. Реализовать функцию подсчета числа вхождений заданного элемента в дерево.
3. \*Изменить функцию добавления элементов для исключения добавления одинаковых символов.
4. \*Оценить сложность процедуры поиска по значению в бинарном дереве.

**Ход работы:**

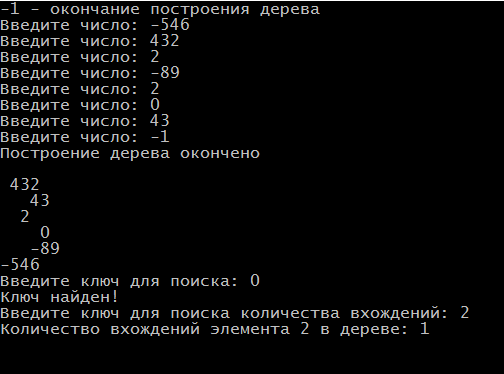
Объявляем структуру, которая представляет узел дерева. Создаем функции для построения, создания, вывода дерева на экран и поиска искомого значения. В главной функции вызываем заданные функции для заполнения, вывода дерева, поиска искомого значения и вывода количества вхождений определенного значения.

**Результаты работы программы:**

Основное задание:



Дополнительно задание:



**Вывод:**

В ходе лабораторной работы нами было изучено бинарное дерево поиска, которое является структурой данных, используемой для хранения и поиска элементов. Реализовали такие функции как: построение, создание, вывод дерева на экран и поиск искомого значения. Оценили сложность кода, она равна O(log(n)).

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <stdlib.h>

#include <string>

#include <stdio.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root;

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

if (root == NULL) {

root = (struct Node\*)malloc(sizeof(struct Node));

root->data = data;

root->left = root->right = NULL;

return root;

}

if (data < root->data) {

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

}

else {

root->right = CreateTree(root->right, 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);

}

struct Node\* search(struct Node\* root, int key) {

if (root == NULL)

return NULL;

if (root == NULL || root->data == key)

return root;

if (key < root->data)

return search(root->left, key);

else

return search(root->right, key);

return NULL;

}

int countOccurrences(struct Node\* root, int data) {

if (root == NULL) return 0;

if (root->data == data) return 1 + countOccurrences(root->left, data) + countOccurrences(root->right, data);

return countOccurrences(root->left, data) + countOccurrences(root->right, data);

}

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, D);

}

print\_tree(root, 0);

int key;

printf("Введите ключ для поиска: ");

scanf("%d", &key);

struct Node\* found = search(root, key);

if (found == NULL) {

printf("Ключ к сожалению не найден\n");

}

else {

printf("Ключ найден!\n");

}

int searchKey;

printf("Введите ключ для поиска количества вхождений: ");

scanf("%d", &searchKey);

int count = countOccurrences(root, searchKey);

printf("Количество вхождений элемента %d в дереве: %d", searchKey, count);

getchar();

getchar();

}

//УЧАСТОК КОДА ДЛЯ 3 ПУНКТА

/\*

struct Node\* search(struct Node\* root, int key) {

if (root == NULL || root->data == key)

return root;

if (key < root->data)

return search(root->left, key);

else

return search(root->right, key);

}

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

if (search(root, data) != NULL)

return root;

if (root == NULL) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

if (data < root->data) {

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

}

else {

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

}

return root;

}\*/