Министерство науки и высшего 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 22ВВВ3:

Байков А. В.

Гераськина Д. А.

Приняли:

Юрова О. В.

Акифьев И. В.

Пенза 2023

**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.

Жирным шрифтом выделено то, что было добавлено к исходному коду, в зависимости от задания

**Задание №1**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

**struct Node {**

**int data; // Данные дерева**

**struct Node\* left; // Левый узел**

**struct Node\* right; // Правый узел**

**};**

// Функция создания дерева

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

{

if (root == NULL) // Если дерево пусто

{

root = (struct Node\*)malloc(sizeof(struct Node)); // Выделяется память

if (root == NULL) // Если выделение памяти невозможно

{

printf("Ошибка выделения памяти");

exit(0); // Завершение работы

}

root->left = NULL;

root->right = NULL;

root->data = data;

return root;

}

// Если элемент больше чем значение текущего узла

if (data > root->data)

{

// Рекурсивный вызов с правым узлом

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

}

else

{

// Рекурсивный вызов с левым узлом

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

}

return root; // Возврат узла

}

// Функция вывода бинарного дерева

void print\_tree(struct Node\* root, int l)

{

// Если дерево пустое

if (root == NULL)

{

return; // То конец

}

// Вывод левого поддерева

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

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

{

printf(" ");

}

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

// Вывод правого поддерева

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

}

**// Функции поиска элемента в бинарном дереве**

**struct Node\* Search(struct Node\* root, int target)**

**{**

**if (root == NULL || root->data == target) // Если дерево пустое или текущей элемент равен искомому**

**{**

**return root; // Указатель на текущей узел**

**}**

**if (target < root->data) // Если искомый элемент меньше текущего**

**{**

**return Search(root->left, target); // Поиск по левому поддереву**

**}**

**else**

**{**

**return Search(root->right, target); // Иначе поиск по правому поддереву**

**}**

**}**

int main()

{

setlocale(LC\_ALL, "RUS");

int D, target, start = 1;

struct Node\* root = NULL;

printf("\t\t\t\tВведите -10, чтобы прекратить построение дерева: \n");

while (start)

{

printf("Введите число: ");

scanf("%d", &D); // Заполнение дерева

if (D == -10) // Если будет введено число -10, заполнение дерева прекратится

{

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

start = 0;

}

else

root = CreateTree(root, D); // Иначе продолжается заполнение дерева

}

print\_tree(root, 0); // Вывод в консоль элементов дерева

**printf("\nКакое число проверить на наличие в дереве?: ");**

**scanf("%d", &target); // Ввод числа для проверки на принадлежность дереву**

**// Вывод о нахождение\отсутствие искомого элемента в дереве**

**struct Node\* result = Search(root, target);**

**if (result != NULL)**

**{**

**printf("\nЗначение %d найдено в дереве\n", target);**

**}**

**else**

**{**

**printf("\nЗначение %d не принадлежит дереву\n", target);**

**}**

return 0;

}

**Задание №2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

**struct Node {**

**int data;**

**struct Node\* left;**

**struct Node\* right;**

**};**

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

{

if (root == NULL)

{

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

if (root == NULL)

{

printf("Ошибка выделения памяти");

exit(0);

}

root->left = NULL;

root->right = NULL;

root->data = data;

return root;

}

if (data > root->data)

{

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

}

else

{

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

}

return root;

}

void print\_tree(struct Node\* root, int l)

{

if (root == NULL)

{

return;

}

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

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

{

printf(" ");

}

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

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

}

**// Функция подсчета количества входов определенного числа в дерево**

**int CountOccurrences(struct Node\* root, int target)**

**{**

**// Если в дереве нет нужного элемента**

**if (root == NULL)**

**{**

**return 0; // Возврат 0**

**}**

**if (root->data == target) // Если нужный элемент найден**

**{**

**// Увеличиваем счётчик и проходимся по правому и левому поддереву**

**return 1 + CountOccurrences(root->left, target) + CountOccurrences(root->right, target);**

**}**

**else if (target < root->data) // Если искомый элемент меньше текущего**

**{**

**return CountOccurrences(root->left, target); // То ищем его в левом поддереве**

**}**

**else // Иначе**

**{**

**return CountOccurrences(root->right, target); // Ищем его в правом поддереве**

**}**

**}**

int main()

{

setlocale(LC\_ALL, "RUS");

int D, start = 1;

struct Node\* root = NULL;

printf("\t\t\t\tВведите -10, чтобы прекратить построение дерева: \n");

while (start)

{

printf("Введите число: ");

scanf("%d", &D);

if (D == -10)

{

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

start = 0;

}

else

root = CreateTree(root, D);

}

print\_tree(root, 0);

**printf("\nВведите значение для подсчета вхождений: ");**

**scanf("%d", &D); // Ввод искомого числа**

**int count = CountOccurrences(root, D); // Счетчик**

**printf("\nЗначение %d встречается %d раз(а) в дереве\n", D, count);**

return 0;

}

**Задание №3**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

**struct Node {**

**int data;**

**struct Node\* left;**

**struct Node\* right;**

**};**

**// Переделанное построение дерева**

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

**{**

**if (root == NULL) // Если дерево пустое**

**{**

**root = (struct Node\*)malloc(sizeof(struct Node)); // Расширение в памяти**

**if (root == NULL) // Если не удалось расширить память**

**{**

**printf("Ошибка выделения памяти");**

**exit(0); // Прекращаем работу**

**}**

**root->left = NULL;**

**root->right = NULL;**

**root->data = data;**

**return root;**

**}**

**if (data < root->data) // Если текущей элемент меньше текущего узла**

**{**

**root->left = CreateTree(root->left, data); // То левое поддерево**

**}**

**else if (data > root->data) // // Если текущей элемент больше текущего узла**

**{**

**root->right = CreateTree(root->right, data); // То правое поддерево**

**} // Игнорируем дублирующиеся значения, которые = узлу**

**return root;**

**}**

int CountOccurrences(struct Node\* root, int target) {

if (root == NULL)

return 0;

if (root->data == target)

return 1 + CountOccurrences(root->left, target) + CountOccurrences(root->right, target);

else if (target < root->data)

return CountOccurrences(root->left, target);

else

return CountOccurrences(root->right, target);

}

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);

}

int main() {

setlocale(LC\_ALL, "RUS");

int end, start = 1;

struct Node\* root = NULL;

printf("\t\t\t\tВведите -10, чтобы прекратить построение дерева: \n");

while (start) {

printf("Введите число: ");

scanf("%d", &end);

if (end == -10) {

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

start = 0;

}

else {

root = CreateTree(root, end);

}

}

print\_tree(root, 0);

return 0;

}

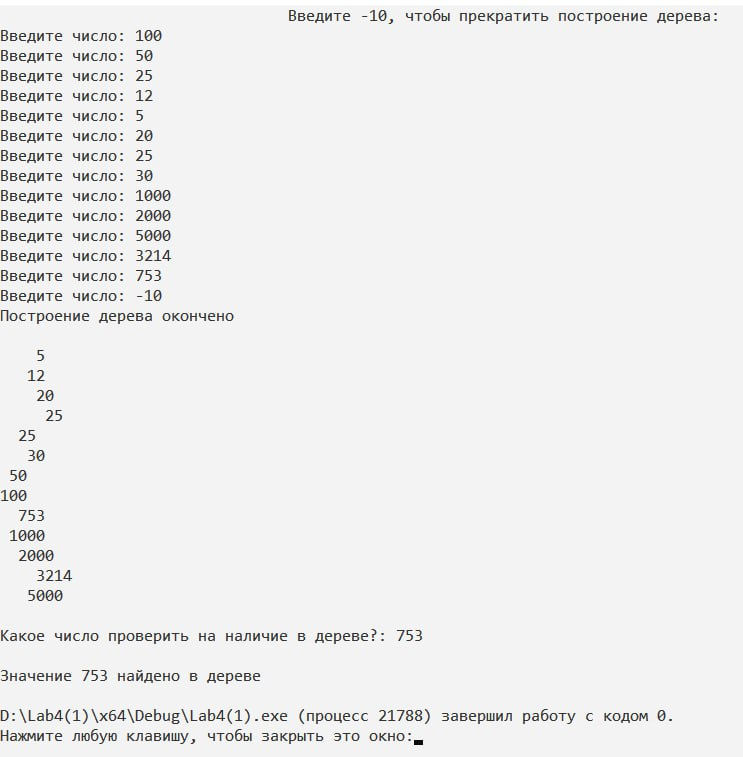
**Задание №4**

Сложность процедуры поиска Search в бинарном дереве поиска зависит от высоты дерева и может быть оценена как O(h), где h - высота дерева.

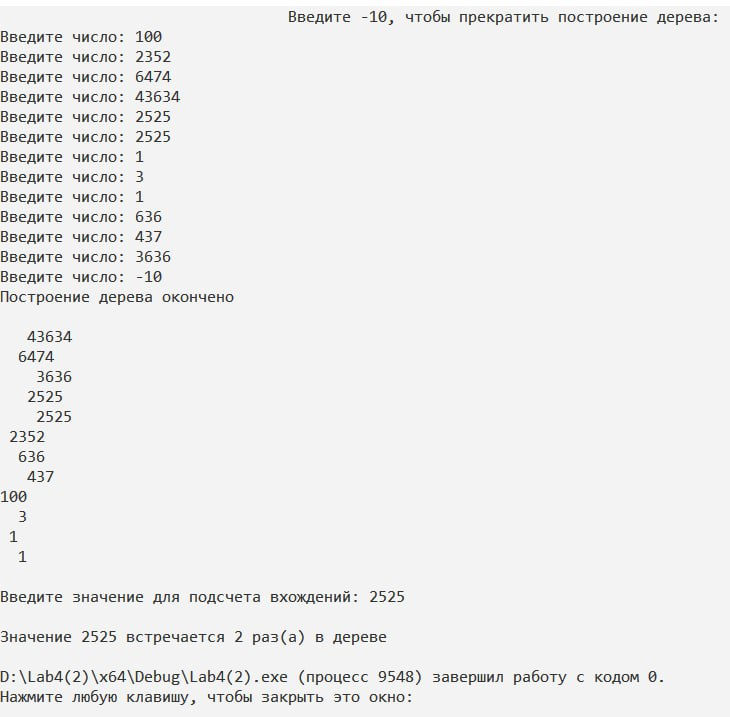
В лучшем случае, когда дерево сбалансировано, высота дерева будет O(logn), где n - количество узлов в дереве.

В наихудшем случае, когда дерево является вырожденным (все узлы идут в одну из ветвей), высота дерева будет равна n, и сложность поиска составит O(n).

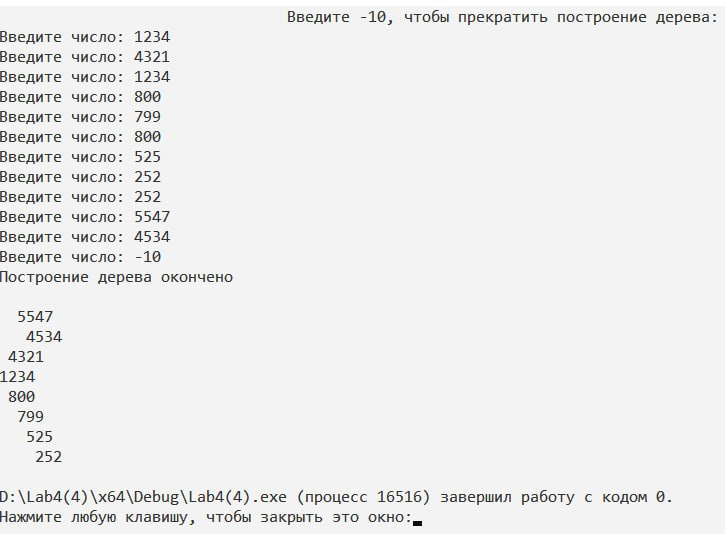
**Результаты работы программ**



**Рисунок 1 - Результат работы программы Lab4.1**



**Рисунок 2 - Результат работы программы Lab4.2**



**Рисунок 3 - Результат работы программы Lab4.3**

**Вывод**

В результате выполнения лабораторной работы были успешно реализованы алгоритмы поиска, подсчета вхождений и добавления элементов в бинарное дерево поиска, а также оценена сложность процедуры поиска.