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 22ВВВ2:

Юртаев Д. Ю.

Шатин Д. Д.

Харитонов А. А.

Приняли:

Акифьев И. В.

Митрохин М. А.

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.

**Задание №1**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root; // указатель на корень дерева

struct Node\* CreateTree(struct Node\* root, struct Node\* r, int data); // создание

void print\_tree(struct Node\* r, int l); // вывод

void SearchData(struct Node\* r, int d); // поиск

int main()

{

// ввод и вывод на русском

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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, root, d);

}

print\_tree(root, 0);

printf("\nВведите число, которое хотите найти: ");

scanf("%d", &d);

SearchData(root, d);

return 0;

}

// функция создания дерева

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

{

if (r == NULL) // если место для узла найдено

{

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; // создание основного корня

// прикрепляем узел к родителю

if (data > root->data) root->left = r;

else root->right = r;

return r;

}

// перемещаемся по дереву с помощью рекурсии для размещения нового узла

if (data > r->data)

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

else

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

return root;

}

// функция вывода дерева

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 SearchData(struct Node\* r, int d)

{

if (d == r->data)

{

printf("Элемент найден!\n");

return;

}

else if (r->left && d > r->data)

{

SearchData(r->left, d);

}

else if (r->right && d < r->data)

{

SearchData(r->right, d);

}

else

{

printf("Элемент не найден!\n");

return;

}

return;

}

**Задание №2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root; // указатель на корень дерева

struct Node\* CreateTree(struct Node\* root, struct Node\* r, int data); // создание

void print\_tree(struct Node\* r, int l); // вывод

int count = 0;

void CountData(struct Node\* r, int d); // подсчет

int main()

{

// ввод и вывод на русском

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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, root, d);

}

print\_tree(root, 0);

printf("\nВведите число, количество вхождений которого хотите узнать: ");

scanf("%d", &d);

CountData(root, d);

return 0;

}

// функция создания дерева

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

{

if (r == NULL) // если место для узла найдено

{

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; // создание основного корня

// прикрепляем узел к родителю

if (data > root->data) root->left = r;

else root->right = r;

return r;

}

// перемещаемся по дереву с помощью рекурсии для размещения нового узла

if (data > r->data)

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

else

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

return root;

}

// функция вывода дерева

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 CountData(struct Node\* r, int d)

{

if (d == r->data)

{

count++;

}

if (r->left && d > r->data)

{

CountData(r->left, d);

}

else if (r->right && d <= r->data)

{

CountData(r->right, d);

}

else

{

printf("Число элементов - %d\n", count);

return;

}

return;

}

**Задание №3**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* root; // указатель на корень дерева

struct Node\* CreateTree(struct Node\* root, struct Node\* r, int data); // создание

void print\_tree(struct Node\* r, int l); // вывод

int main()

{

// ввод и вывод на русском

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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, root, d);

}

print\_tree(root, 0);

return 0;

}

// функция создания дерева

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

{

if (r == NULL) // если место для узла найдено

{

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; // создание основного корня

// прикрепляем узел к родителю

if (data > root->data) root->left = r;

else root->right = r;

return r;

}

// перемещаемся по дереву с помощью рекурсии для размещения нового узла

if (data == r->data) // проверка на присутствие элемента в дереве

{

printf("\n>> Такой элемент уже есть\n\n");

return r;

}

else if (data > r->data)

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

else

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

return root;

}

// функция вывода дерева

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);

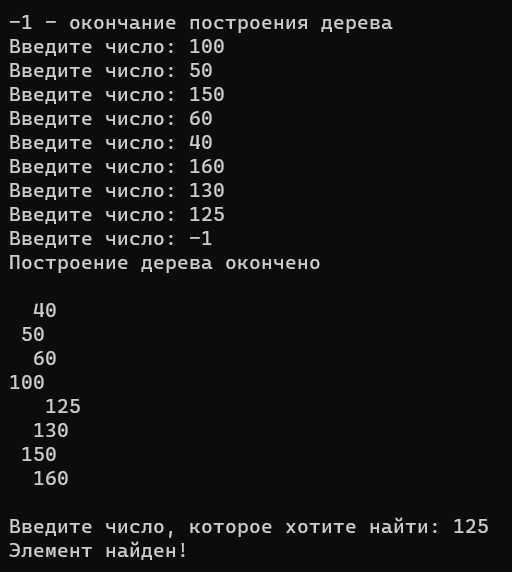
}

**Задание №4**

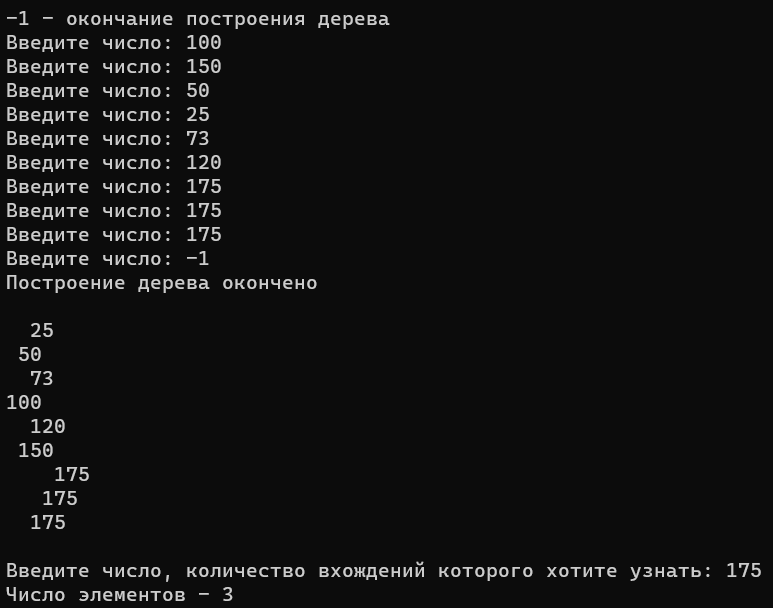
В лучшем случае, когда дерево сбалансировано, сложность алгоритма поиска элемента дерева будет O(logn), где n - количество узлов в дереве.

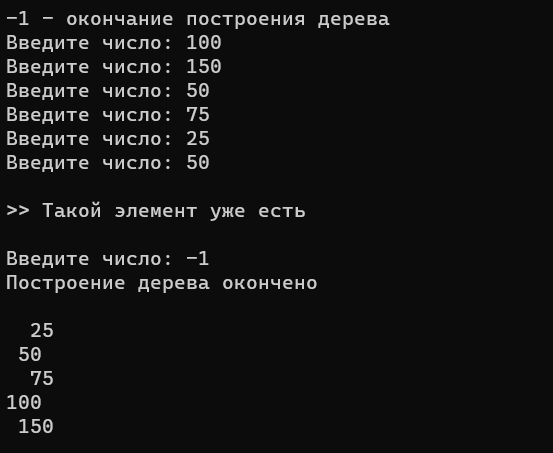
В наихудшем случае, когда дерево является вырожденным (все узлы идут в одну из ветвей), высота дерева будет равна n, и сложность поиска составит O(n).

**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.