Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №4

по курсу «Логика и основы алгоритмизации в ИЗ»

на тему: «Бинарное дерево поиска»

Выполнил:

студент группы 23ВВВ4

Соснин Глеб

Проверил:

доцент, Юрова О. В.

Пенза, 2024

**Цель** - научиться создавать бинарное дерево поиска, пользоваться функциями бинарного дерева поиска.

**Задание 1:** Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.

**Листинг программы:**

#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, struct Node\* r, int data)

{

if (r == NULL)

{

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

if (r == NULL)

{

printf("Ошибка выделения памяти\n");

exit(0);

}

r->left = NULL;

r->right = NULL;

r->data = data;

if (root == NULL) return r;

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

else root->right = r;

return r;

}

if (data > r->data)

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

else

CreateTree(r, r->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);

}

// Функция поиска элемента в дереве

struct Node\* SearchTree(struct Node\* r, int data)

{

if (r == NULL || r->data == data)

return r;

if (data > r->data)

return SearchTree(r->left, data);

else

return SearchTree(r->right, data);

}

int main()

{

setlocale(LC\_ALL, "");

struct Node\* root = NULL;

int D, start = 1;

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);

}

}

printf("Ваше дерево:\n");

print\_tree(root, 0);

// Поиск элемента

printf("Введите число для поиска в дереве: ");

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

struct Node\* searchResult = SearchTree(root, D);

if (searchResult != NULL)

printf("Элемент %d найден в дереве.\n", D);

else

printf("Элемент %d не найден в дереве.\n", D);

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

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

return 0;

}

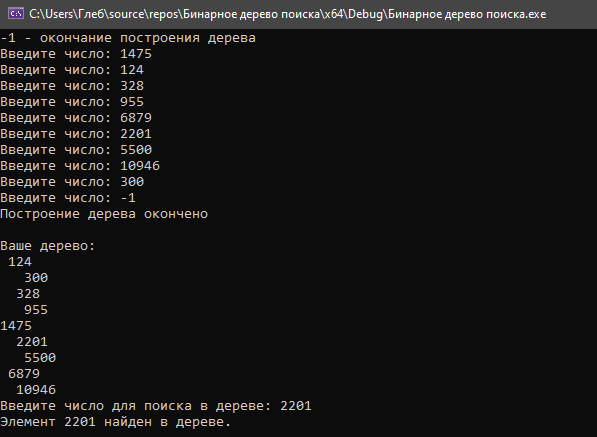


Рисунок 1 – Поиск числа в бинарном дереве

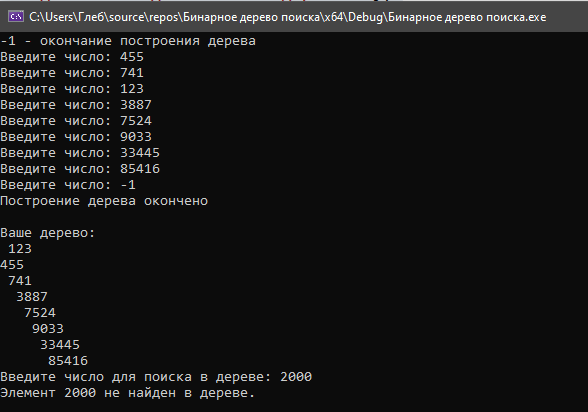


Рисунок 2 – Поиск числа в бинарном дереве

Функция поиска возвращает указатель на найденный элемент или *NULL*, если элемент не найден.

**Задание 2:** Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

**Листинг программы:**

#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, struct Node\* r, int data)

{

if (r == NULL)

{

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

if (r == NULL)

{

printf("Ошибка выделения памяти\n");

exit(0);

}

r->left = NULL;

r->right = NULL;

r->data = data;

if (root == NULL) return r;

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

else root->right = r;

return r;

}

if (data > r->data)

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

else

CreateTree(r, r->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);

}

// Функция поиска элемента в дереве

struct Node\* SearchTree(struct Node\* r, int data)

{

if (r == NULL || r->data == data)

return r;

if (data > r->data)

return SearchTree(r->left, data);

else

return SearchTree(r->right, data);

}

// Функция подсчета числа вхождений элемента в дереве

int CountOccurrences(struct Node\* r, int data)

{

if (r == NULL)

return 0;

int count = (r->data == data) ? 1 : 0;

return count + CountOccurrences(r->left, data) + CountOccurrences(r->right, data);

}

int main()

{

setlocale(LC\_ALL, "");

struct Node\* root = NULL;

int D, start = 1;

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);

}

}

printf("Ваше дерево:\n");

print\_tree(root, 0);

// Поиск элемента

printf("Введите число для поиска в дереве: ");

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

struct Node\* searchResult = SearchTree(root, D);

if (searchResult != NULL)

printf("Элемент %d найден в дереве.\n", D);

else

printf("Элемент %d не найден в дереве.\n", D);

printf("Введите число для подсчета вхождений в дереве: ");

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

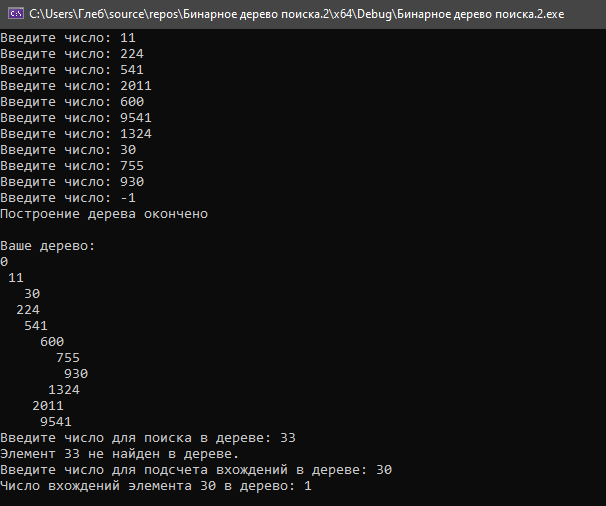
int count = CountOccurrences(root, D);

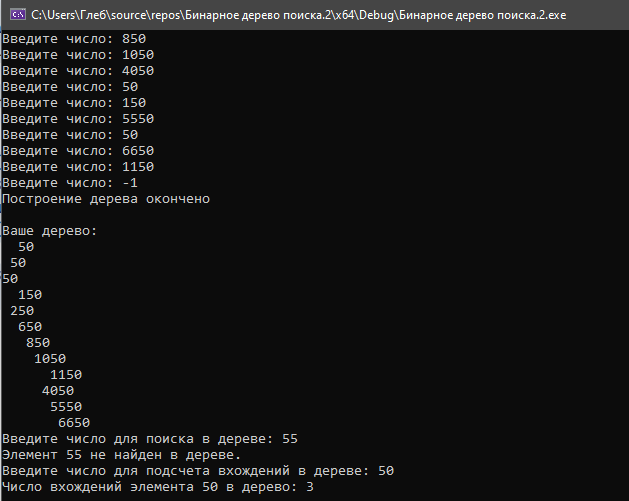
printf("Число вхождений элемента %d в дерево: %d\n", D, count);

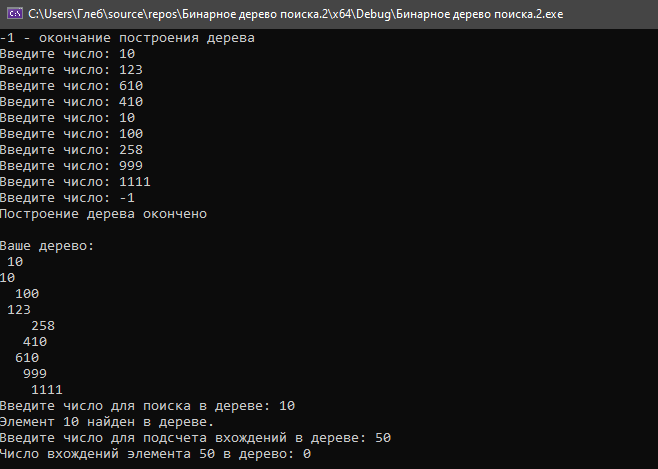
scanf\_s("%d", &D);

return 0;

}







Рисунки 3, 4, 5 – Подсчет числа вхождения заданного элемента в дерево

Для подсчета количества вхождений заданного элемента нужно пройти все узлы дерева и подсчитать все совпадения.

Программа корректно ищет элементы и подсчитывает количество их вхождений в бинарном дереве.

**Задание 3:** Изменить функцию добавления элементов для исключения добавления одинаковых символов.

**Листинг программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left, \* right;

};

// Функция создания нового узла

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->left = r;

else root->right = r;

return r;

}

if (data == r->data)

{

printf("Элемент с значением %d уже существует в дереве.\n", data);

return root;

}

if (data > r->data)

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

else

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

return root;

}

// Функция для 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);

}

int main()

{

setlocale(LC\_ALL, "");

int D, start = 1;

struct Node\* 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("Нажмите любую клавишу для выхода...");

getchar();

getchar();

return 0;

}

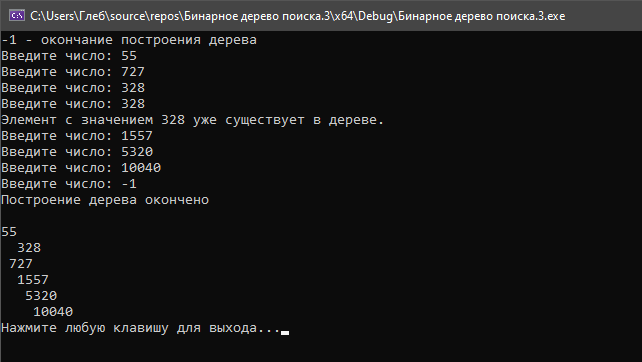


Рисунок 6 – Поиск одинаковых чисел в бинарном дереве

Программа строит бинарное дерево и исключает добавление одинаковых значений. Если элемент уже существует в дереве, выводится соответствующее сообщение.

**Задание 4:** Оценить сложность процедуры поиска по значению в бинарном дереве.

**Листинг программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

struct Node {

int data;

struct Node\* left, \* 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->left = r;

else root->right = r;

return r;

}

if (data > r->data)

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

else

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

return root;

}

// Функция для поиска элемента в дереве

struct Node\* Find(struct Node\* r, int data)

{

if (r == NULL)

{

printf("Элемент не найден\n");

return NULL;

}

if (data == r->data)

{

printf("Элемент %d найден\n", data);

return r;

}

if (data > r->data)

return Find(r->left, data);

else

return Find(r->right, data);

}

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, "");

int D, start = 1;

struct Node\* root = NULL;

struct Node\* found = 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);

// Поиск элемента

printf("Введите элемент для поиска: ");

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

found = Find(root, D);

printf("Нажмите любую клавишу для выхода...");

getchar();

getchar();

return 0;

}

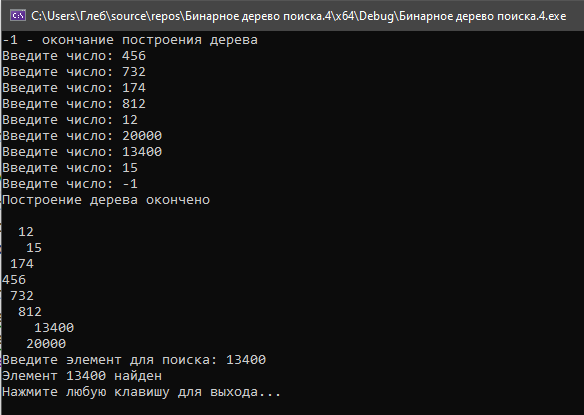


Рисунок 7 – Поиск элемента в бинарном дереве

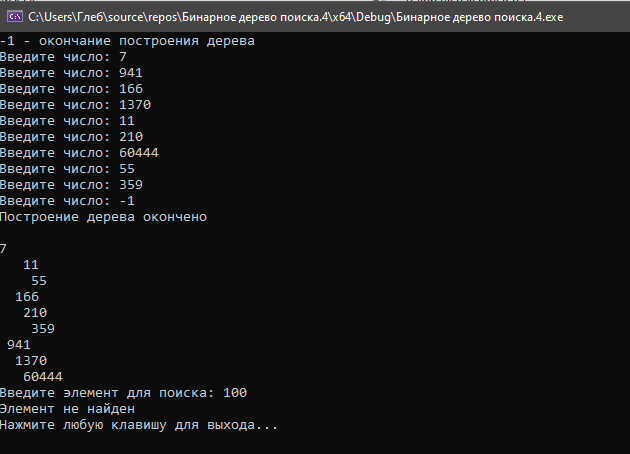


Рисунок 8 – Поиск элемента в бинарном дереве

Введённая функция *Find* осуществляет поиск элемента и выводит сообщение о нахождении или отсутствии искомого значения.

Оценка сложности поиска:

Лучший случай *(O(log n))*: Если дерево сбалансировано, количество узлов, которые нужно проверить, сокращается вдвое на каждом уровне.

Худший случай *(O(n))*: Если дерево не сбалансировано (например, вставка элементов по порядку), дерево превращается в линейную структуру, и поиск выполняется линейно.

**Вывод:** научились создавать бинарное дерево поиска, пользоваться функциями бинарного дерева поиска.