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**

**студенты группы 23ВВВ2:**

Родионов А. А.

**Приняли:**

Митрохин М. А.

Юрова О. В.

Пенза 2024

**Цель работы:** Изучить структуру и основные операции бинарного дерева поиска, реализовать алгоритмы поиска и модификации дерева на языке C, а также проанализировать эффективность поиска в бинарном дереве.

**Задание 1:**

Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.

**Задание 2:**

Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.

**\*Задание 3:**

Изменить функцию добавления элементов для исключения добавления одинаковых символов.

**\*Задание 4:**

Оценить сложность процедуры поиска по значению в бинарном дереве.

**Ход работы**

**Задание 1**

**Описание программы**

Программа реализует бинарное дерево поиска и предоставляет функциональность для создания дерева, его вывода и поиска значений в нем. Она позволяет пользователю создать дерево, визуализировать его структуру и выполнять поиск значений в построенном дереве. Программа работает следующим образом:

1. Структура Node:  
   Определяет узел дерева, содержащий данные (int data) и указатели на левого и правого потомков.
2. Функция CreateTree:
   * Создает новый узел, если текущий узел (r) пустой.
   * Если корень (root) пустой, новый узел становится корнем.
   * Иначе, новый узел добавляется как левый или правый потомок в зависимости от значения.
   * Рекурсивно вызывается для добавления узлов глубже в дерево.
3. Функция print\_tree:
   * Рекурсивно выводит дерево в повернутом виде (правое поддерево сверху, левое снизу).
   * Использует отступы для визуализации уровней дерева.
4. Функция search:
   * Рекурсивно ищет заданное значение в дереве.
   * Возвращает указатель на найденный узел или NULL, если значение не найдено.
5. Функция main:
   * Позволяет пользователю вводить числа для создания дерева.
   * Выводит построенное дерево.
   * Предоставляет интерфейс для поиска значений в дереве.

Листинг программы смотрите в Приложении А.

**Результат работы программы**

На рисунке приведён результат работы программы №1.

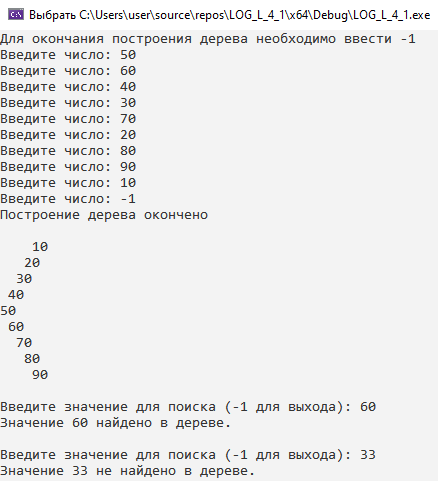


Рисунок 1 – результат работы программы №1

**Задание 2**

**Описание программы**

Программа для данного задания основана на программе для задания 1. В функцию search() было добавлено дополнительное условие, в котором подсчитывается число вхождений заданного элемента в дерево.

if (root->data == value)

{

(\*count)++;

search(root->left, value, count);

search(root->right, value, count);

return 1;

}

Листинг программы смотрите в Приложении В.

**Результат работы программы**

На рисунке 2 приведён результат работы программы №2.

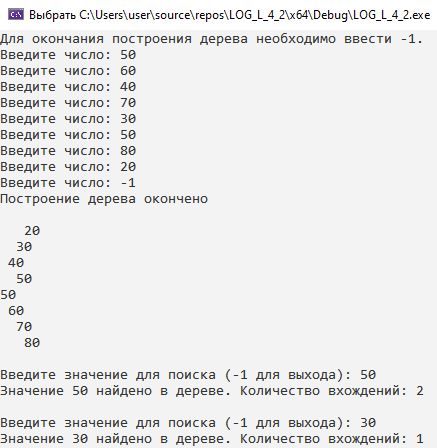


Рисунок 2 – результат работы программы №2

**\*Задание 3**

**Описание программы**

Программа для данного задания основана на программе для задания 1.

Была изменена функция CreateTree():

1. Когда функция находит подходящее место для вставки нового узла (когда r == NULL), она создает новый узел и заполняет его данными.
2. Затем происходит проверка, является ли этот узел корнем дерева (if (root == NULL) return r;). Если да, то узел просто возвращается как новый корень.
3. Если узел не является корнем, то происходит сравнение значения нового узла со значением текущего корня:

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

else if (data < root->data) root->right = r;

else {

free(r);

return root;

}

1. Основной момент - это блок else. Он срабатывает, когда значение нового узла равно значению текущего корня (data == root->data). В этом случае:
   * Освобождается память, выделенная для нового узла (free(r)).
   * Возвращается текущий корень без изменений (return root).
2. Таким образом, если попытаться добавить значение, которое уже существует в дереве, новый узел не будет добавлен, а существующая структура дерева останется без изменений.
3. В рекурсивной части функции также проверяется равенство значений, и если значения равны, рекурсия просто прекращается без добавления нового узла.

Этот алгоритм гарантирует, что в дереве не будет повторяющихся значений, так как при попытке добавить уже существующее значение, оно просто игнорируется.

Листинг программы смотрите в Приложении С.

**Результат работы программы**

На рисунке 3 приведён результат работы программы №3.

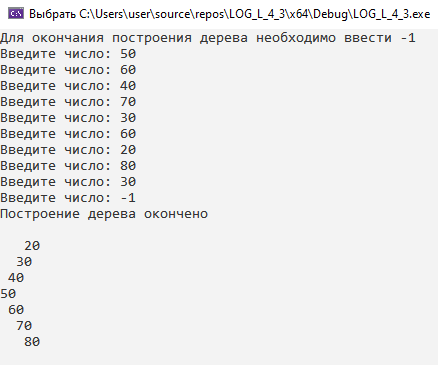


Рисунок 3 – результат работы программы №3

**\*Задание 4**

**Оценка сложности процедуры поиска по значению**

Поиск по значению осуществляется при помощи функции search():

struct Node\* search(struct Node\* root, int value)

{

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

return root;

if (value > root->data)

return search(root->left, value);

return search(root->right, value);

}

Функция search() использует рекурсивный подход для поиска значения в бинарном дереве поиска. В худшем случае, функция может пройти от корня до самого глубокого листа дерева.

Сложность этой функции зависит от высоты дерева:

1. В лучшем случае (когда искомое значение находится в корне), сложность будет O(1).
2. В среднем случае для сбалансированного бинарного дерева поиска, высота дерева будет примерно log(n), где n - количество узлов. Таким образом, средняя сложность будет O(log n).
3. В худшем случае, когда дерево вырождается в линейную структуру (например, когда все узлы добавляются в одном направлении), высота дерева может быть равна n, и сложность будет O(n).

Поскольку в О-символике обычно рассматривается наихудший случай, сложность функции search() можно оценить как O(h), где h - высота дерева.

Итак, общая оценка сложности функции search():

* O(h), где h - высота дерева
* O(log n) для сбалансированного дерева
* O(n) в худшем случае для несбалансированного дерева

**Выводы**

Были изучены структуры и основные операции бинарного дерева поиска, реализованы алгоритмы поиска и модификации дерева, а также была проанализирована эффективность поиска в бинарном дереве.

**Приложение А**

**Листинг программы для задания 1**

#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\* 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->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);

}

struct Node\* search(struct Node\* root, int value)

{

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

return root;

if (value > root->data)

return search(root->left, value);

return search(root->right, value);

}

int main()

{

setlocale(LC\_ALL, "Russian");

int D, start = 1;

root = NULL;

printf("Для окончания построения дерева необходимо ввести -1\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);

while (1)

{

printf("\nВведите значение для поиска (-1 для выхода): ");

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

if (D == -1)

break;

struct Node\* result = search(root, D);

if (result != NULL)

printf("Значение %d найдено в дереве.\n", D);

else

printf("Значение %d не найдено в дереве.\n", D);

}

return 0;

}

**Приложение B**

**Листинг программы для задания 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\* 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->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);

}

int search(struct Node\* root, int value, int\* count)

{

if (root == NULL)

return 0;

if (root->data == value)

{

(\*count)++;

search(root->left, value, count);

search(root->right, value, count);

return 1;

}

if (value > root->data)

return search(root->left, value, count);

return search(root->right, value, count);

}

int main()

{

setlocale(LC\_ALL, "Russian");

int D, start = 1;

root = NULL;

printf("Для окончания построения дерева необходимо ввести -1.\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);

while (1)

{

printf("\nВведите значение для поиска (-1 для выхода): ");

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

if (D == -1)

break;

int count = 0;

int found = search(root, D, &count);

if (found)

printf("Значение %d найдено в дереве. Количество вхождений: %d\n", D, count);

else

printf("Значение %d не найдено в дереве.\n", D);

}

return 0;

}

**Приложение С**

**Листинг программы для задания 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;

};

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->left = r;

else if (data < root->data) root->right = r;

else {

free(r);

return root;

}

return r;

}

if (data > r->data)

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

else if (data < r->data)

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);

}

struct Node\* search(struct Node\* root, int value)

{

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

return root;

if (value > root->data)

return search(root->left, value);

return search(root->right, value);

}

int main() {

setlocale(LC\_ALL, "Russian");

int D, start = 1;

root = NULL;

printf("Для окончания построения дерева необходимо ввести -1\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);

while (1)

{

printf("\nВведите значение для поиска (-1 для выхода): ");

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

if (D == -1)

break;

struct Node\* result = search(root, D);

if (result != NULL)

printf("Значение %d найдено в дереве.\n", D);

else

printf("Значение %d не найдено в дереве.\n", D);

}

return 0;

}