МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Логика и ОА в ИЗ»

на тему «Бинарное дерево поиска»

Выполнили:

Студенты группы 23ВВВ2

Севостьянов Олег

Кандаков Андрей

Принял:

Митрохин М.А.

Юрова О.В.

Пенза 2024

**Цель работы**

Изучение бинарного дерева поиска.

**Лабораторное задание**

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.

**Пояснение к коду**

Задание 1/2:

**Структура данных Node**

* **data** — хранит значение узла дерева (целое число).
* **left** — указатель на левый дочерний узел (если он есть).
* **right** — указатель на правый дочерний узел (если он есть).
* Это структура для создания бинарного дерева, где каждый узел может иметь два потомка.

**Функция для создания узла в дереве CreateTree**

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

* Функция создает новый узел с заданным значением и вставляет его в дерево.
* **root** — указатель на корень дерева или поддерева, куда будет вставлен новый узел.
* **data** — значение, которое будет храниться в новом узле.

if (root == NULL) {

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

if (root == NULL) {

printf("Ошибка выделения памяти\n");

exit(0);

}

root->left = NULL;

root->right = NULL;

root->data = data;

return root;

}

* Если root равен NULL, это значит, что дерево пустое, и нужно создать новый узел:
  + Выделяется память для нового узла.
  + Если память не выделена (ошибка), выводится сообщение и программа завершает выполнение.
  + Инициализируются указатели на левого и правого потомков как NULL.
  + Присваивается значение узла, и новый узел возвращается.

**Функция вывода дерева print\_tree**

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

if (r == NULL) {

return;

}

* Функция для печати дерева в виде структуры, отображающей иерархию.
* Если узел пустой (NULL), функция завершается.

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

* Сначала рекурсивно вызывается функция для правого поддерева. Параметр l + 1 увеличивает уровень отступа, так как правое поддерево будет располагаться ниже и правее.

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

printf(" ");

}

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

* Отображается текущий узел на уровне отступа l. Каждый уровень увеличивает отступ, чтобы отобразить дерево сверху вниз.

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

}

* Рекурсивный вызов для левого поддерева, с тем же уровнем отступа, увеличенным на 1.
* **Функция поиска узла search**

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

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

return root;

}

* Функция поиска узла по значению.
* Если узел пустой (NULL) или значение узла совпадает с ключом (key), возвращается текущий узел.

if (key < root->data) {

return search(root->left, key);

}

* Если ключ меньше значения текущего узла, поиск продолжается в левом поддереве.
* return search(root->right, key);
* }
* Если ключ больше значения текущего узла, поиск продолжается в правом поддереве.
* **Функция подсчета вхождений элемента в дерево countOccurrences**

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

if (root == NULL) {

return 0;

}

* Функция подсчета количества вхождений заданного значения в дереве.
* Если узел пустой, возвращается 0 (элемента нет).

int count = 0;

if (root->data == key) {

count = 1;

}

* Если значение текущего узла совпадает с ключом, увеличивается счетчик.
* count += countOccurrences(root->left, key);
* count += countOccurrences(root->right, key);
* Рекурсивно вызывается подсчет для левого и правого поддеревьев, добавляя их результаты к общему счету.
* return count;
* }
* Возвращается общее количество вхождений ключа в дереве.
* **Главная функция main**

int main() {

setlocale(LC\_ALL, "");

* Устанавливает локаль, чтобы поддерживать ввод/вывод на русском языке (если это необходимо).

int D, start = 1;

struct Node\* root = NULL;

* Переменная D для хранения вводимых значений.
* start — флаг, управляющий циклом ввода данных.
* root — указатель на корень дерева, инициализируется как NULL.

printf("-1 - окончание построения дерева\n");

while (start) {

printf("Введите число: ");

scanf("%d", &D);

if (D == -1) {

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

start = 0;

}

else {

root = CreateTree(root, D);

}

}

* Цикл для ввода данных для построения дерева.
* Если вводится -1, цикл завершается, и выводится сообщение об окончании построения дерева.
* Вводимые данные передаются в функцию CreateTree для вставки в дерево.

printf("Вывод дерева:\n");

print\_tree(root, 0);

* Вывод дерева с помощью функции print\_tree.

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

scanf("%d", &D);

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

if (foundNode != NULL) {

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

}

else {

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

}

* Ввод значения для поиска в дереве.
* Используется функция search для поиска элемента в дереве.
* Если элемент найден, выводится сообщение о его наличии, иначе — сообщение об отсутствии.
* int occurrences = countOccurrences(root, D);
* printf("Число вхождений элемента %d в дерево: %d\n", D, occurrences);
* Подсчитывается количество вхождений элемента в дереве с помощью функции countOccurrences.

Задание 3\*:

if (data < root->data) {

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

}

else if (data > root->data) {

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

}

* Если данные уже есть в узле (то есть data == root->data), то узел не создается, так как в дереве не будет дублирующих элементов.
* Если значение меньше текущего узла, рекурсивно вызывается функция для левого поддерева.
* Если значение больше, рекурсивно вызывается функция для правого поддерева.

**Результат работы программы**

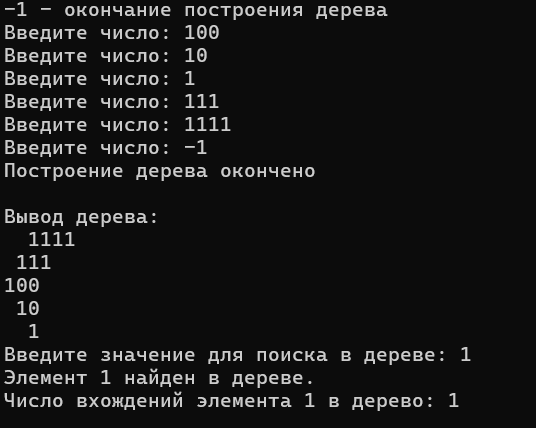


Рисунок 1 – результат работы программы (Задание 1)

**Вывод**

В ходе работы изучено бинарное дерево поиска.