**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **«Деревья»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Аршин А.Д |  |
| Преподаватель | Манирагена В. |  |

Санкт-Петербург

2024

**Введение**

**Задание вариант 21:**

. Написать и отладить программу для работы с деревьями по предложенному преподавателем варианту индивидуального задания (табл. П.2.2). Программа должна выводить на экран изображение дерева с разметкой его вершин, сделанной заданным способом, а под ним — последовательность меток вершин при обходе дерева и результат вычисления заданного параметра. Можно взять за основу учебный пример, убрав из него всё лишнее.

2. Сделать узел дерева и дерево в целом объектами соответствующих классов, а обходы дерева — функциями-членами для класса «дерево».

3. Объявить в классе «дерево» деструктор и все конструкторы, поддерживаемые по умолчанию. Сделать невозможным использование тех конструкторов, которые на самом деле не нужны. Сделать в тексте программы временные дополнения и убедиться, что это действительно так.

**Обход внутренний, двоичное дерево, разметка прямая, найти Высоту правого поддерева для корня**

**Цель работы**

Цель работы заключается в разработке и реализации программы для создания и визуализации бинарного дерева, а также выполнения базовых операций над ним, таких как добавление узлов, внутренний (инфиксный) обход и вычисление высоты поддеревьев. Визуализация структуры дерева должна выполняться в текстовой сетке, а результаты обхода и вычисления высоты выводиться на экран.

**Постановка задачи**

**Задача состоит в следующем:**

1. Создать класс для представления узлов бинарного дерева. Каждый узел должен содержать значение и указатели на левого и правого детей.

2. Реализовать бинарное дерево с функциями:

- добавления узлов по заданному значению;

- визуализации структуры дерева в виде текстовой сетки;

- выполнения внутреннего (инфиксного) обхода дерева;

- вычисления высоты правого поддерева.

3. Визуализировать дерево в сетке размером 10x60, где каждый уровень дерева представлен на отдельной строке.

4. Выполнить внутренний обход дерева и вывести значения узлов в порядке возрастания.

5. Определить и вывести высоту правого поддерева относительно корня дерева.

**Описание решения**

**Решение включает разработку двух классов:**

**1.** TreeNode — класс, представляющий узел бинарного дерева. Каждый узел содержит значение (`char value`), указатель на левого ребенка (`TreeNode left`) и указатель на правого ребенка (`TreeNode right`). Конструктор класса инициализирует эти поля.

2. BinaryTree — класс для представления бинарного дерева. В этом классе реализованы следующие методы:

- addNode() — добавление узла с заданным значением. Для добавления используется рекурсивная функция addNodeRecursive(), которая вставляет значение в дерево согласно правилам бинарного поиска.

- printTree() — метод визуализации дерева. Для этого используется двумерный массив символов (`grid`), который отображает структуру дерева. Метод fillGrid() заполняет сетку символами, соответствующими узлам дерева.

- inOrderTraversal() — метод для выполнения внутреннего обхода дерева. Он использует рекурсивную функцию inOrderTraversalRecursive(), которая обходит левое поддерево, выводит текущее значение узла и затем обходит правое поддерево.

- getRightSubtreeHeight() — метод для вычисления высоты правого поддерева. Высота рассчитывается с помощью рекурсивной функции getHeight().

В конце работы программа выводит:

1. Визуализированную структуру дерева в сетке.

2. Результат внутреннего обхода дерева.

3. Высоту правого поддерева.

**Контрольные примеры**

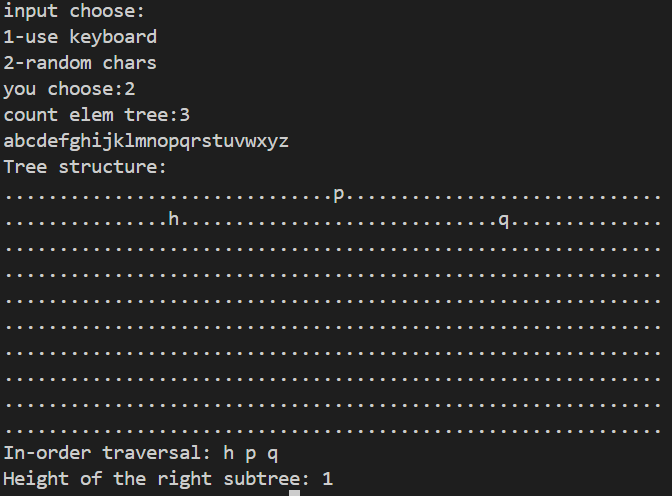
**Примеры выполнения программы**

**Пример 1(ввод с клавиатуры)**

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

**Пример 2(рандомный ввод автоматом)**



**Контрольные вопросы**

1. Чем отличаются алгоритмы для разных способов обхода деревьев?

Алгоритмы обхода деревьев различаются порядком посещения узлов:

- Прямой (pre-order): сначала обрабатывается корневой узел, затем левое поддерево и правое поддерево. Порядок: корень → левый ребенок → правый ребенок.

- Внутренний (in-order): сначала обрабатывается левое поддерево, затем корневой узел, потом правое поддерево. Порядок: левый ребенок → корень → правый ребенок. Этот метод подходит для бинарных деревьев поиска, так как он выводит элементы в отсортированном порядке.

- Обратный (post-order): сначала обрабатываются оба поддерева (левое и правое), а потом корневой узел. Порядок: левый ребенок → правый ребенок → корень.

- Обход в ширину: посещаются все узлы на одном уровне дерева, а затем переходят к следующему уровню, начиная с корня.

Каждый из этих алгоритмов применяется в зависимости от задачи: поиск, сортировка или другие операции с деревьями.

2. Нужно ли сочетать ввод данных для построения дерева с клавиатуры с его обходом?

Не обязательно. Эти операции могут быть независимыми. Ввод данных можно реализовать на этапе построения дерева, а сам обход выполнять отдельно после того, как дерево построено. Однако в некоторых задачах может потребоваться ввод данных и немедленное выполнение обхода (например, если узлы добавляются динамически, и мы сразу хотим видеть изменения в дереве).

3. Можно ли считать применённые вами алгоритмы обхода дерева эффективными?

Да, алгоритмы, реализованные в коде (инфиксный обход и вычисление высоты дерева), считаются эффективными для большинства задач с бинарными деревьями. Время выполнения рекурсивных алгоритмов обхода — это \(O(n)\), где \(n\) — количество узлов в дереве. Каждый узел посещается один раз, что делает их достаточно производительными для стандартных бинарных деревьев.

4. Нужно ли создавать отдельные классы для узла и для дерева в целом, или можно ограничиться одним универсальным, рассматривая любой узел как корень некоторого поддерева?

Создание отдельных классов для узла и дерева — хороший объектно-ориентированный подход, так как он делает код более понятным и структурированным. Узел представляет базовый элемент дерева, а дерево управляет узлами и предоставляет методы для их обработки (например, добавление узлов, обходы и т.д.). Однако, если дерево имеет простую структуру, можно ограничиться одним классом, где каждый узел будет рассматриваться как корень поддерева. Это подходит для небольших программ, но в более сложных задачах отдельные классы улучшают читаемость и расширяемость кода.

**Заключение:**

В ходе выполнения задания была разработана программа для построения и обхода бинарного дерева с использованием объектно-ориентированного подхода. Были реализованы основные алгоритмы работы с деревом, включая добавление узлов, внутренний (инфиксный) обход и вычисление высоты правого поддерева. Программа продемонстрировала возможность графического отображения дерева в консоли с использованием двумерной сетки.

Благодаря раздельному созданию классов для узлов и дерева, код стал более структурированным и понятным. Такой подход позволяет легко модифицировать и расширять функциональность программы.

Кроме того, реализованный инфиксный обход дерева показал свою эффективность для бинарного дерева поиска, выводя значения узлов в отсортированном порядке. Рассмотренные алгоритмы обладают оптимальной сложностью O(n), что делает их пригодными для применения в задачах с бинарными деревьями.

Таким образом, задание продемонстрировало важность правильного выбора алгоритмов обхода и структуры данных для решения различных задач с использованием бинарных деревьев.