《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 大二 软工 | 姓名 | 丁彦君 | 学号 | 2329403126 |
| 实验布置日期 | | 2024/11/05 | | 提交  日期 | 2024/11/26 | | 成绩 |  |

课程实践实验5：二叉树操作

## 一、问题描述及要求

**本实验要求设计并实现一个通用的二叉树操作系统，完成以下功能：**

**1.创建空二叉树**

**2.根据前序和中序序列构建二叉树**

**3.判断二叉树是否为空**

**4.实现先序、中序、后序、层次遍历（递归与非递归）**

**5.统计节点数量、叶子节点数量和二叉树高度**

**6.计算二叉树的宽度**

**7.插入节点**

**8.判断二叉树是否为镜像对称**

**9.复制二叉树（拷贝构造和赋值重载）**

## 概要设计

**（1）对实验内容的理解。**

**本实验是对数据结构与算法中二叉树概念的综合运用。通过构建二叉树并实现其基本操作，深入掌握二叉树的遍历、节点插入、树结构计算和特性判断等内容。**

**（2）系统的功能列表**

**以下为系统的功能列表：**

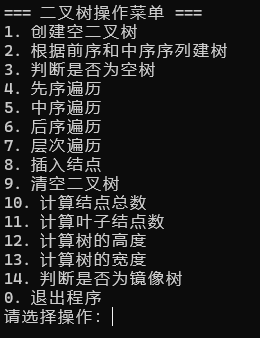
**基本操作：创建空树、根据序列构建二叉树**

**遍历操作：递归与非递归的先序、中序、后序、层次遍历**

**属性计算：节点总数、叶子节点数量、高度、宽度**

**高级操作：节点插入、镜像判断、树的拷贝与赋值**

**（3）程序运行的界面设计**

****

**（4）总体设计思路**

**数据结构：采用链表结构表示二叉树，节点类型为 TreeNode，定义如下：**

**struct TreeNode {**

**int val;**

**TreeNode\* left;**

**TreeNode\* right;**

**TreeNode(int x) : val(x), left(NULL), right(NULL) {}**

**};**

**类设计：类 BinaryTree 用于实现二叉树的所有操作，主要方法包括：**

**基本功能：createTree、buildTree**

**遍历功能：preOrder、inOrder、postOrder、levelOrder、preOrder1、inOrder1、postOrder1**

**高级功能：insertNode、isMirror**

**属性计算：countNode、countLeave、height、width**

**类之间的关系**

**该实验仅设计一个核心类 BinaryTree，其功能通过成员方法实现。**

**（5）程序结构设计**

**程序文件**

**main.cpp：主程序文件，包含菜单与用户交互**

**BinaryTree.h：二叉树类的头文件**

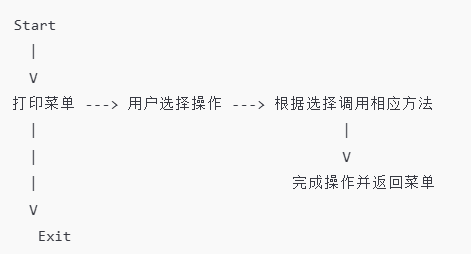
**BinaryTree.cpp：二叉树类的实现文件**

**模块关系**

**主程序通过实例化 BinaryTree 类对象调用其方法，完成所有功能。**

## 三、详细设计

**1. 主函数流程图**



**2. 关键算法设计**

**(1) 根据前序和中序序列构建二叉树**

**(2) 计算二叉树宽度**

## 四、实验结果

## 

## 五、实验分析与探讨

**测试结果的分析**

**测试策略：**

**本实验通过设计多个测试用例，涵盖了所有功能点，包括二叉树的构建、遍历、插入、节点统计、树的高度与宽度计算等。测试过程分为以下几个步骤：**

**首先验证基本功能是否正确，例如空树创建、判空操作等。**

**然后依次测试各种遍历方法，分别验证递归与非递归实现是否输出一致。**

**最后测试复杂功能，例如插入节点后更新结构的正确性，宽度和高度计算的准确性。**

**性能分析：**

**构建二叉树：构建操作时间复杂度为 O(n)，其中 n 为节点数量。空间复杂度受递归栈限制，为 O(h)，其中 h 为树的高度。**

**遍历：递归和非递归遍历时间复杂度均为 O(n)。递归方法的空间复杂度为 O(h)，非递归方法为 O(n)（栈的存储开销）。**

**宽度计算：层次遍历中使用队列存储节点，时间复杂度为 O(n)，空间复杂度受队列大小限制，为 O(w)，其中 w 为最大宽度。**

**高度计算：递归求高度时间复杂度为 O(n)，每个节点访问一次，空间复杂度为 O(h)。**

**结论：通过测试，所有功能均正常运行，输出符合预期。算法在效率与实现难度上达到了平衡，但针对大规模数据集，递归方法可能受到栈溢出的限制。**

**遇到的问题与解决方法**

**1.构建树时的输入验证问题：**

**前序和中序序列输入错误（如包含不存在的节点或重复节点）可能导致程序崩溃。**

**解决：**

**添加输入验证逻辑，确保前序和中序序列仅包含唯一字符且长度一致。**

**遍历算法的边界条件处理**

**2.问题：**

**树为空时，遍历函数调用可能出现意外行为。**

**解决：**

**在每个遍历方法中添加判空逻辑，避免空指针异常。**

**非递归遍历的栈操作逻辑**

**3.问题：**

**非递归后序遍历实现逻辑较复杂，容易导致重复访问或遗漏节点。**

**解决：**

**通过双指针（当前节点和上一次访问节点）优化逻辑，确保遍历顺序正确。**

**内存管理问题**

**4.问题：**

**二叉树的动态分配内存未正确释放，导致潜在的内存泄漏。**

**解决：**

**实现析构函数和清理方法，确保在对象销毁时释放所有内存。**

**宽度计算效率问题**

**5.问题：**

**初始实现中使用递归方法计算宽度，导致较高的时间复杂度。**

**解决：**

**改用层次遍历方法，通过队列优化计算效率。罗列出来，并写出解决问题的方法。**

## 六、小结

**本次实验全面实现了二叉树的构建、遍历、统计等核心功能，涵盖递归与非递归实现方式，解决了节点插入、序列构建及层次遍历等多个难点，同时验证了算法的正确性与运行效率。通过多组测试分析了程序在不同输入场景下的表现，归纳了算法的时间和空间复杂度，为进一步优化树的功能（如增加搜索与平衡操作）奠定了基础。此外，实验过程总结了问题解决的经验，为后续设计更复杂数据结构提供了实践指导。**

## 附录：源代码

**1、实验环境：Dev-c++**

**2、代码：见附件**