实验报告

数据结构 *Data Structures*

报告标题: **树算法**
学号: 19240212
姓名: 华博文
日期: 2025 年 11 月 11 日

一 实验环境

1 操作系统

Ubuntu 24.04.3 LTS x86_64，Linux 6.14.0-28-generic，AMD Ryzen 7 7700 (16) @ 5.3GHz。

2 编程工具

NeoVim v0.12.0-dev，gcc 13.3.0。

3 其他工具

LaTeX (TeXstudio)。

二 实验内容及其完成情况

1 实验目的

1. 掌握树的基本概念和存储结构；
2. 理解树的遍历方式及其应用；
3. 掌握树常用操作的算法实现；
4. 培养使用 C++ 实现复杂数据结构的能力。

2 实验内容

编写程序，实现树类及若干应用算法。要求采用孩子兄弟表示法，实现以下功能：

1. 构造函数：根据序偶集合构造对象；
2. 析构函数：释放所有结点空间（本实验采用智能指针自动管理）；
3. 先根、后根遍历算法；
4. 计算每个结点的度；
5. 计算树的高度；
6. 输出根结点到每个叶子结点的路径。

三　实现细节与过程

本节简要说明实现的关键点与算法契约（输入 / 输出 / 错误处理）。

1　设计契约

- 输入：结点关系的序偶集合（如 std::vector<std::pair<**int**, **int**>>），每个序偶表示父子关系；
- 输出：支持遍历、度、高度、路径等操作的树结构；
- 错误模式：输入非法（如环、孤立结点）可通过异常或提示处理。

2　数据结构定义

采用孩子兄弟表示法，核心结点类型如下：

```
template <typename T> struct TreeNode {
    T data;
    std::unique_ptr<TreeNode<T>> firstChild;
    std::unique_ptr<TreeNode<T>> nextSibling;
    TreeNode(const T &val) : data(val) {}
};
```

树类接口如下：

```
template <typename T> class Tree {
  public:
    Tree(const std::vector<std::pair<T, T>> &pairs);
    void preOrder() const;
    void postOrder() const;
    void printDegrees() const;
    int getHeight() const;
    void printPaths() const;
    const TreeNode<T> *getRoot() const;
};
```

3　树的构建与析构

构造函数根据序偶集合自动建立树结构，采用哈希表辅助查找，智能指针自动释放所有结点空间，无需手动析构。

4 遍历与操作实现

- 先根遍历：先访问根，再递归访问所有孩子；
- 后根遍历：递归访问所有孩子，最后访问根；
- 结点度：统计每个结点的孩子数；
- 树高度：递归计算所有孩子的最大高度；
- 路径输出：递归遍历，遇到叶子结点时输出路径。

5 测试用例与运行结果

设计用例如下：

```
std::vector<std::pair<int, int>> pairs = {
    {1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6}, {3, 7}
};
Tree<int> tree(pairs);

std::cout << " 先根遍历： ";
tree.preOrder();  // 1 2 4 5 3 6 7

std::cout << " 后根遍历： ";
tree.postOrder(); // 4 5 2 6 7 3 1

std::cout << " 每个结点的度： " << std::endl;
tree.printDegrees();
// 结点 1 的度： 2
// 结点 2 的度： 2
// 结点 4 的度： 0
// 结点 5 的度： 0
// 结点 3 的度： 2
// 结点 6 的度： 0
// 结点 7 的度： 0

std::cout << " 树的高度： " << tree.getHeight() << std::endl; // 3

std::cout << " 根到每个叶子的路径： " << std::endl;
tree.printPaths();
// 路径： 1 -> 2 -> 4
// 路径： 1 -> 2 -> 5
// 路径： 1 -> 3 -> 6
// 路径： 1 -> 3 -> 7
```

实际运行结果与预期一致，所有功能均通过测试。

四　复杂度分析与讨论

对主要操作给出时间 / 空间复杂度总结：

- 构造树：$\mathcal{O}(n)$，每个结点和边只处理一次。
- 遍历（先根 / 后根）：$\mathcal{O}(n)$，每个结点访问一次。
- 计算度：$\mathcal{O}(n)$，每个结点的孩子只需遍历一次。
- 计算高度：$\mathcal{O}(n)$，递归访问所有结点。
- 输出路径：$\mathcal{O}(n)$，每条路径只访问一次所有结点。

其中 n 为结点总数。空间复杂度最坏也是 $\mathcal{O}(n)$（递归栈或辅助结构）。

如果树是平衡的，递归栈深度为 $\mathcal{O}(\log n)$，退化为链时为 $\mathcal{O}(n)$。

五　实验总结

本次实验实现了基于孩子兄弟表示法的树结构，支持多种遍历和常用操作，代码结构清晰，注释完整，测试用例覆盖全面，满足实验要求。采用现代 C++ 编程风格，便于扩展和维护。