AVL 树删除操作实现报告

1. 基础删除逻辑

AVL 树中的删除操作基本上与标准二叉搜索树的删除类似，主要包括以下几种情况：

- **无子节点**：直接删除该节点。
- **一个子节点**：将该节点替换为其唯一的子节点，然后删除。
- **两个子节点**：找到**右子树的最小节点**，将该节点的值替换到当前节点，然后递归删除右子树中的最小节点。

2. 维持平衡性

在删除节点后，树的平衡性可能会受到影响，因此我们在删除操作中引入了平衡调整：

- `height`:给节点的结构里增加了height
- `updateHeight`：更新节点的高度信息，确保平衡检查的准确性。
- `balance`：检查每个节点的左右子树高度差是否超过允许的最大不平衡值（1）。如果超过，则根据结构进行旋转操作。
  - `rotateWithLeftChild`：左单旋转
  - `rotateWithRightChild`：左单旋转
  - `doubleWithLeftChild`：左右双旋转
  - `doubleWithRightChild`：右左双旋转

3. 平衡调整流程

1. 删除操作完成后，逐层回溯调用 `updateHeight`，更新节点的高度。
2. 调用 `balance` 检查当前节点的平衡性。如果左右子树的高度差超过允许的不平衡值，`balance` 函数会根据不平衡的类型选择适当的旋转操作以恢复平衡。

4.特别的

为了避免不必要的数据拷贝操作，我们使用了智能指针 `std::unique_ptr` 来管理节点中的数据。在替换节点时，通过交换指针来减少数据拷贝，从而提高性能。