第四章总结：树

总结

树的基本结构

- 树是一种递归定义的数据结构，用于表示具有层次关系的元素。
- 二叉树是树的基本形式，每个节点最多有两个子节点。

二叉搜索树（BST）

- 左子树的所有节点值小于根节点，右子树的所有节点值大于根节点。
- 常见操作：插入、删除、查找。
- 平均时间复杂度为 $O(\log n)$，最坏情况为 $O(n)$。

平衡树

为避免 BST 退化成链表，提出了以下平衡树：

- **AVL 树**：严格平衡的二叉搜索树，通过旋转操作保持平衡。
- **Splay 树**：访问节点时进行旋转，将节点移到树根，优化后续访问。
- **B 树**：适合外部存储的大型数据集管理。

树的遍历方式

- **前序遍历**：根节点 $\rightarrow$ 左子树 $\rightarrow$ 右子树。
- **中序遍历**：左子树 $\rightarrow$ 根节点 $\rightarrow$ 右子树。
- **后序遍历**：左子树 $\rightarrow$ 右子树 $\rightarrow$ 根节点。
- **层序遍历**：按层次逐层访问节点。

集合和映射

- C++ 提供了 `set` 和 `map` 容器，分别用于存储集合和键值对。
- 它们基于红黑树实现，操作复杂度为 $O(\log n)$。

应用场景

- 树广泛用于表达式解析、文件系统、数据库索引等领域。
- 平衡树（如 AVL 树和 B 树）适用于需要高效查询的大型数据集。

学习重点

- 理解树的基本结构和性质。
- 掌握二叉搜索树及平衡树的操作与应用场景。
- 熟悉树的遍历算法及其实现。
- 了解 `set` 和 `map` 容器的使用及其实现原理。