7.1 二叉树

二叉树（binary tree）是一种非线性数据结构，代表"祖先"与"后代"之间的派生关系，体现了"一分为二"的分治逻辑。与链表类似，二叉树的基本单元是节点，每个节点包含值、左子节点引用和右子节点引用。

Python

```python
class TreeNode:
    """二叉树节点类"""
    def __init__(self, val: int):
        self.val: int = val                # 节点值
        self.left: TreeNode | None = None  # 左子节点引用
        self.right: TreeNode | None = None # 右子节点引用
```

每个节点都有两个引用（指针），分别指向左子节点（left-child node）和右子节点（right-child node），该节点被称为这两个子节点的父节点（parent node）。当给定一个二叉树的节点时，我们将该节点的左子节点及其以下节点形成的树称为该节点的左子树（left subtree），同理可得右子树（right subtree）。

在二叉树中，除叶节点外，其他所有节点都包含子节点和非空子树。如图 7-1 所示，如果将"节点 2"视为父节点，则其左子节点和右子节点分别是"节点 4"和"节点 5"，左子树是"节点 4 及其以下节点形成的树"，右子树是"节点 5 及其以下节点形成的树"。

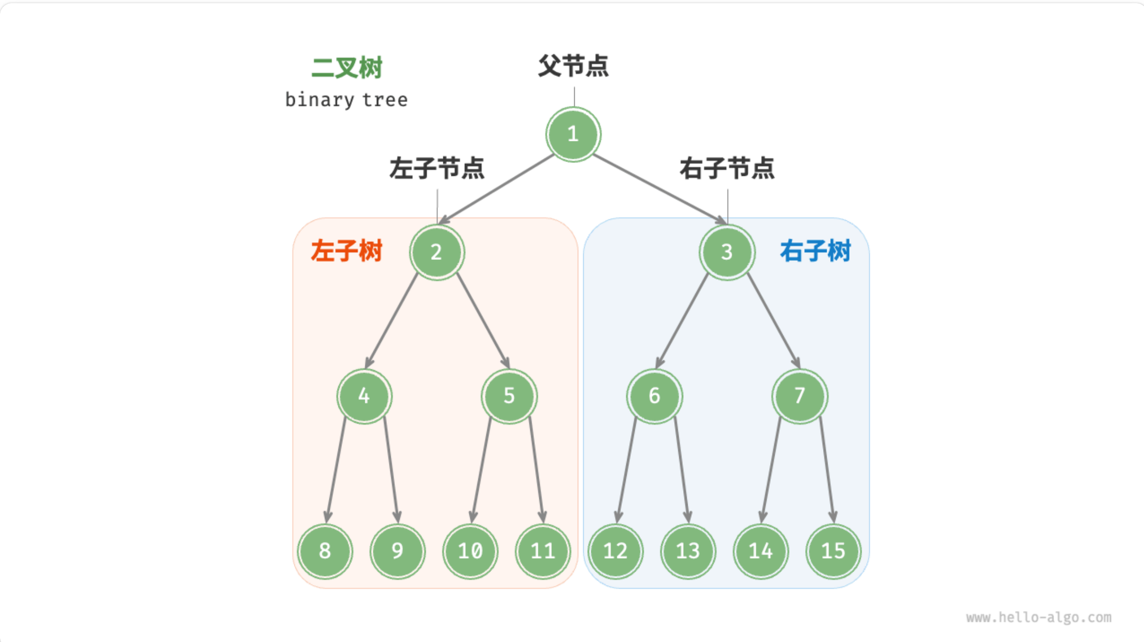

图 7-1 父节点、子节点、子树

7.1.1 二叉树常见术语

二叉树的常用术语如图 7-2 所示。

- 根节点（root node）：位于二叉树顶层的节点，没有父节点。
- 叶节点（leaf node）：没有子节点的节点，其两个指针均指向 `None` 。
- 边（edge）：连接两个节点的线段，即节点引用（指针）。
- 节点所在的层（level）：从顶至底递增，根节点所在层为 1 。
- 节点的度（degree）：节点的子节点的数量。在二叉树中，度的取值范围是 0、1、2 。
- 二叉树的高度（height）：从根节点到最远叶节点所经过的边的数量。
- 节点的深度（depth）：从根节点到该节点所经过的边的数量。
- 节点的高度（height）：从距离该节点最远的叶节点到该节点所经过的边的数量。

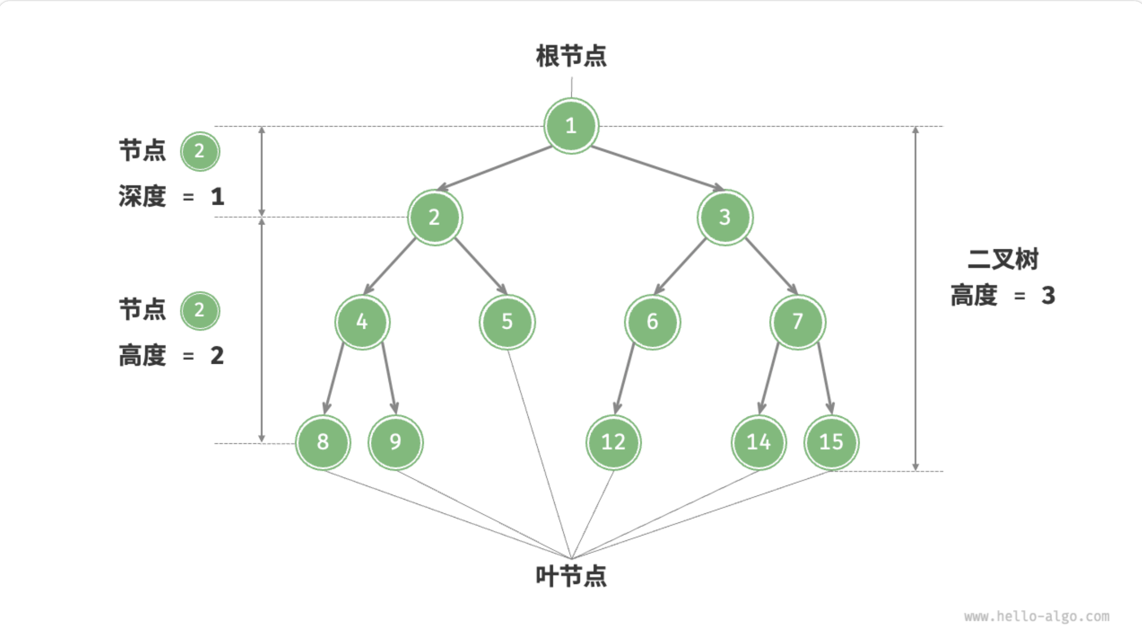

图 7-2 二叉树的常用术语

> **Tip**
>
> 请注意，我们通常将“高度”和“深度”定义为“经过的边的数量”，但有些题目或教材可能会将其定义为“经过的节点的数量”。在这种情况下，高度和深度都需要加 1 。

7.1.2 二叉树基本操作

1. 初始化二叉树

与链表类似，首先初始化节点，然后构建引用（指针）。

Python

binary_tree.py

```
# 初始化二叉树
# 初始化节点
n1 = TreeNode(val=1)
n2 = TreeNode(val=2)
n3 = TreeNode(val=3)
n4 = TreeNode(val=4)
n5 = TreeNode(val=5)
# 构建节点之间的引用（指针）
n1.left = n2
n1.right = n3
n2.left = n4
n2.right = n5
```

2. 插入与删除节点

与链表类似，在二叉树中插入与删除节点可以通过修改指针来实现。图 7-3 给出了一个示例。

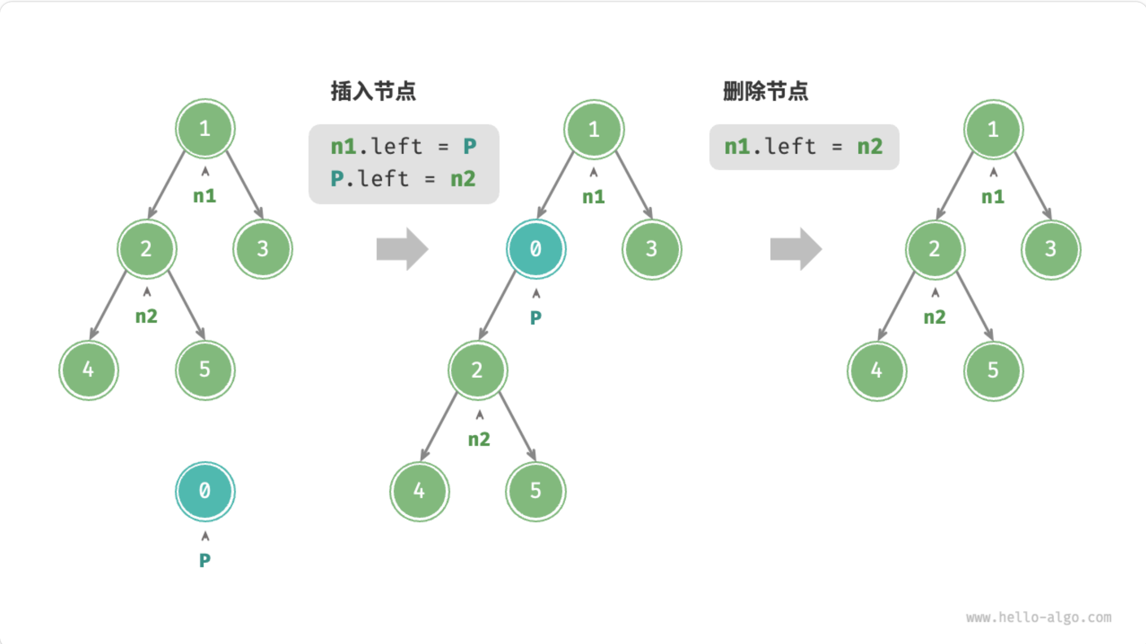

图 7-3 在二叉树中插入与删除节点

Python

binary_tree.py

```
# 插入与删除节点
p = TreeNode(0)
# 在 n1 -> n2 中间插入节点 P
n1.left = p
p.left = n2
# 删除节点 P
n1.left = n2
```

Tip

需要注意的是，插入节点可能会改变二叉树的原有逻辑结构，而删除节点通常意味着删除该节点及其所有子树。因此，在二叉树中，插入与删除通常是由一套操作配合完成的，以实现有实际意义的操作。

7.1.3 常见二叉树类型

1. 完美二叉树

如图 7-4 所示，完美二叉树（perfect binary tree）所有层的节点都被完全填满。在完美二叉树中，叶节点的度为 0 ，其余所有节点的度都为 2 ；若树的高度为 h ，则节点总数为 $2^{h+1}-1$ ，呈现标准的指数级关系，反映了自然界中常见的细胞分裂现象。

Tip

请注意，在中文社区中，完美二叉树常被称为满二叉树。

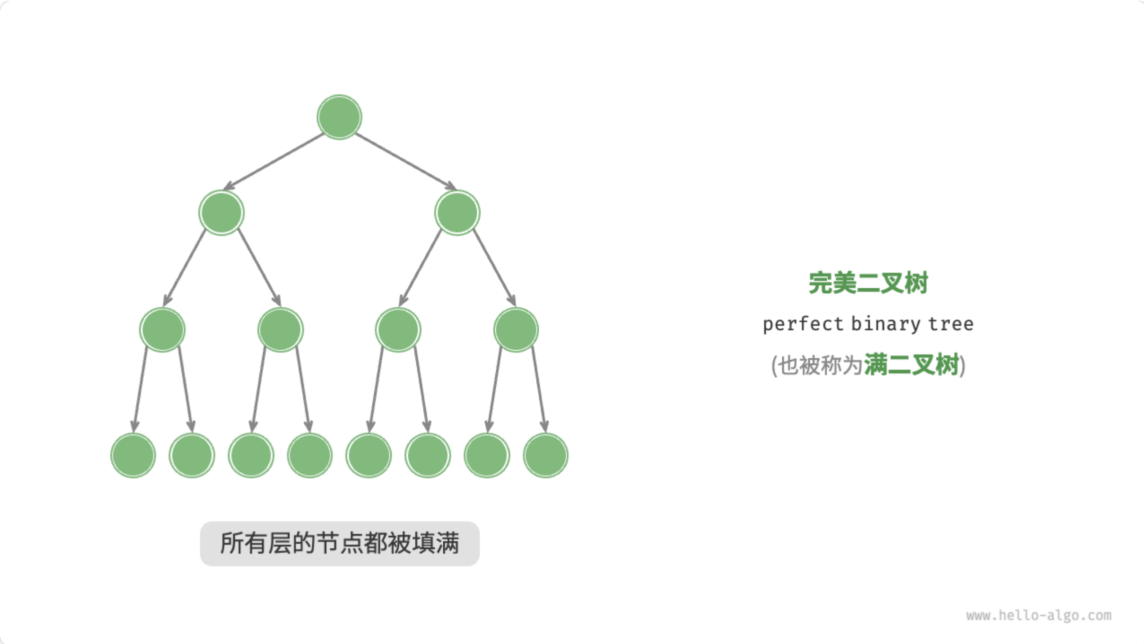

图 7-4 完美二叉树

2. 完全二叉树

如图 7-5 所示，完全二叉树（complete binary tree）只有最底层的节点未被填满，且最底层节点尽量靠左填充。

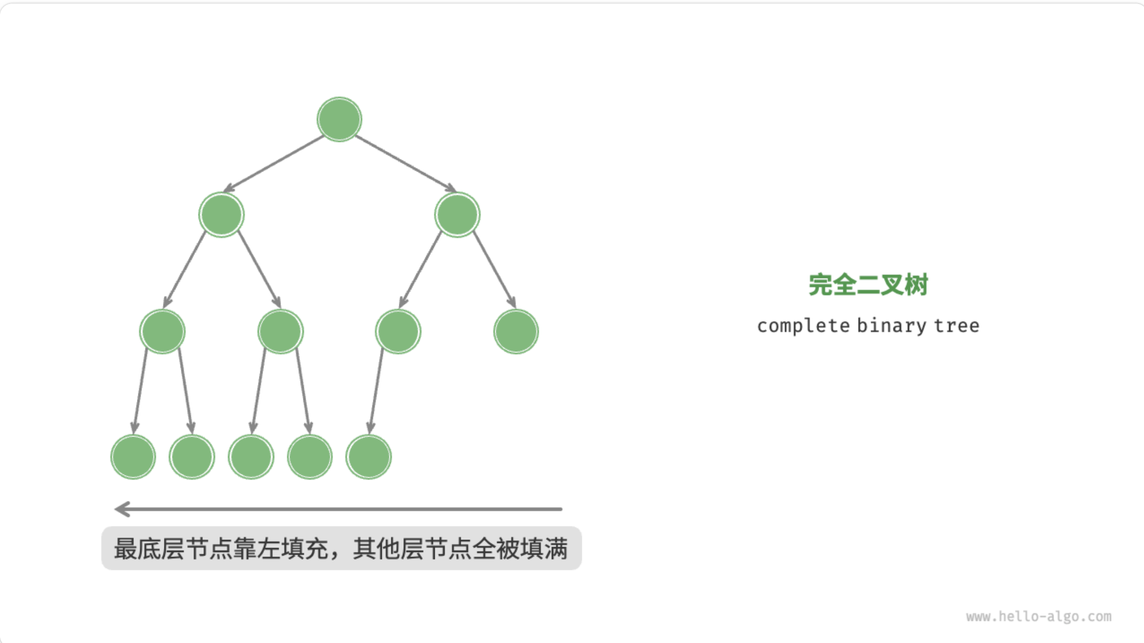

图 7-5 完全二叉树

3. 完满二叉树

如图 7-6 所示，完满二叉树（full binary tree）除了叶节点之外，其余所有节点都有两个子节点。

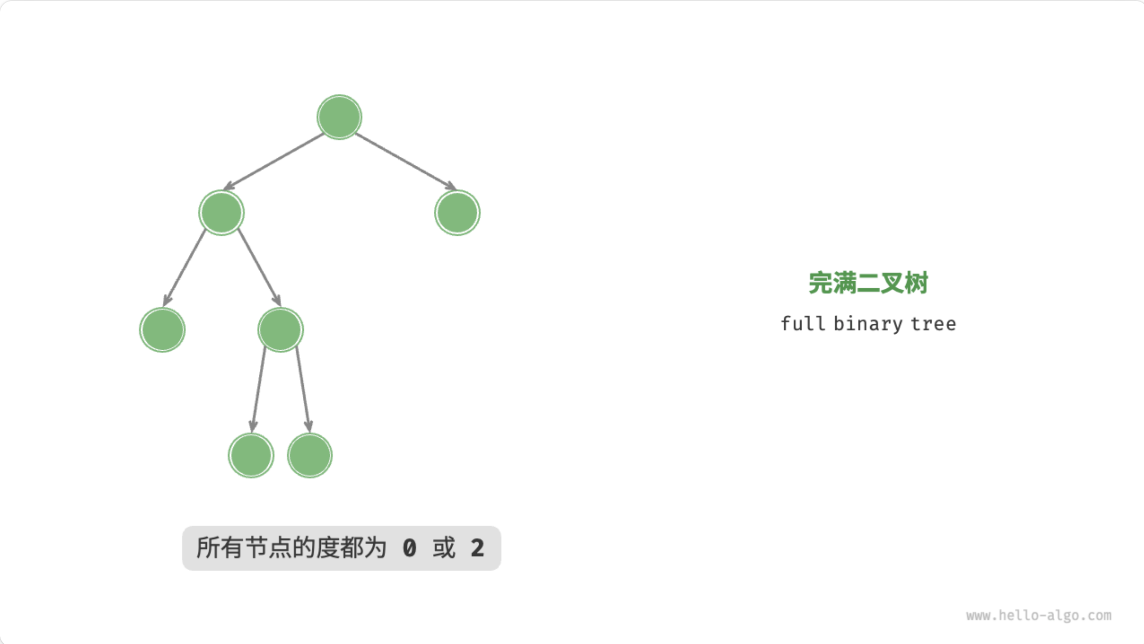

图 7-6 完满二叉树

4. 平衡二叉树

如图 7-7 所示，平衡二叉树（balanced binary tree）中任意节点的左子树和右子树的高度之差的绝对值不超过 1 。

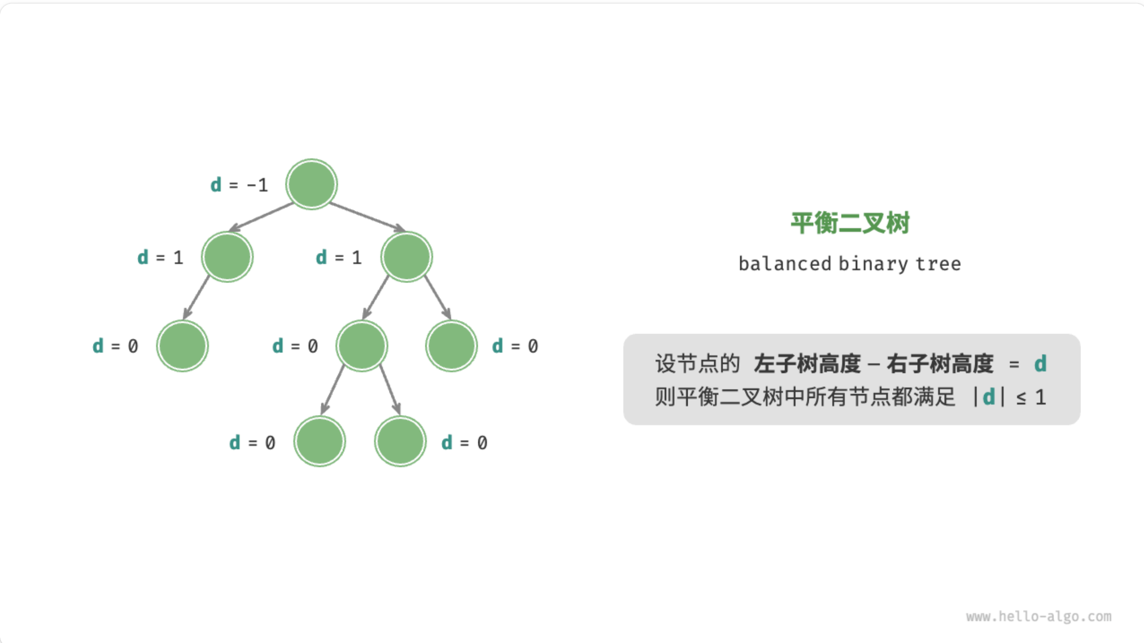

图 7-7 平衡二叉树

7.1.4 二叉树的退化

图 7-8 展示了二叉树的理想结构与退化结构。当二叉树的每层节点都被填满时，达到“完美二叉树”；而当所有节点都偏向一侧时，二叉树退化为“链表”。

- 完美二叉树是理想情况，可以充分发挥二叉树“分治”的优势。
- 链表则是另一个极端，各项操作都变为线性操作，时间复杂度退化至 $O(n)$ 。

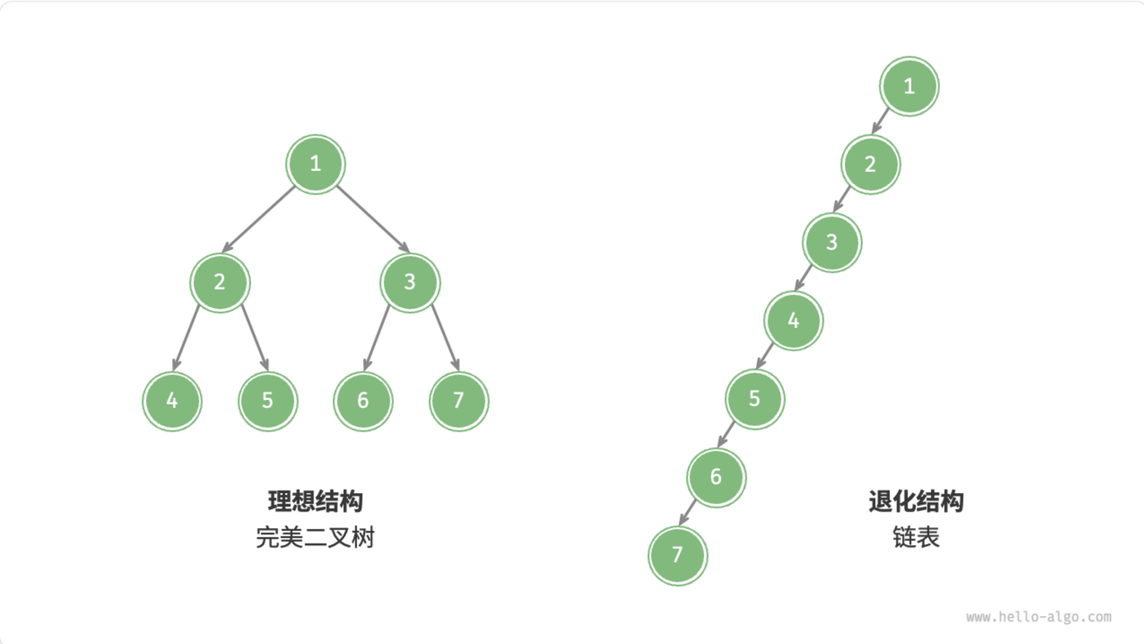

图 7-8 二叉树的最佳结构与最差结构

如表 7-1 所示，在最佳结构和最差结构下，二叉树的叶节点数量、节点总数、高度等达到极大值或极小值。

表 7-1 二叉树的最佳结构与最差结构

	完美二叉树	**链表**
第 i 层的节点数量	2^{i-1}	1
高度为 h 的树的叶节点数量	2^h	1
高度为 h 的树的节点总数	$2^{h+1}-1$	$h+1$
节点总数为 n 的树的高度	$\log_2(n+1)-1$	$n-1$