

## 7.2 二叉树遍历

从物理结构的角度来看，树是一种基于链表的数据结构，因此其遍历方式是通过指针逐个访问节点。然而，树是一种非线性数据结构，这使得遍历树比遍历链表更加复杂，需要借助搜索算法来实现。

二叉树常见的遍历方式包括层序遍历、前序遍历、中序遍历和后序遍历等。

### 7.2.1 层序遍历

如图 7-9 所示，层序遍历（level-order traversal）从顶部到底部逐层遍历二叉树，并在每一层按照从左到右的顺序访问节点。

层序遍历本质上属于广度优先遍历（breadth-first traversal），也称广度优先搜索（breadth-first search, BFS），它体现了一种“一圈一圈向外扩展”的逐层遍历方式。

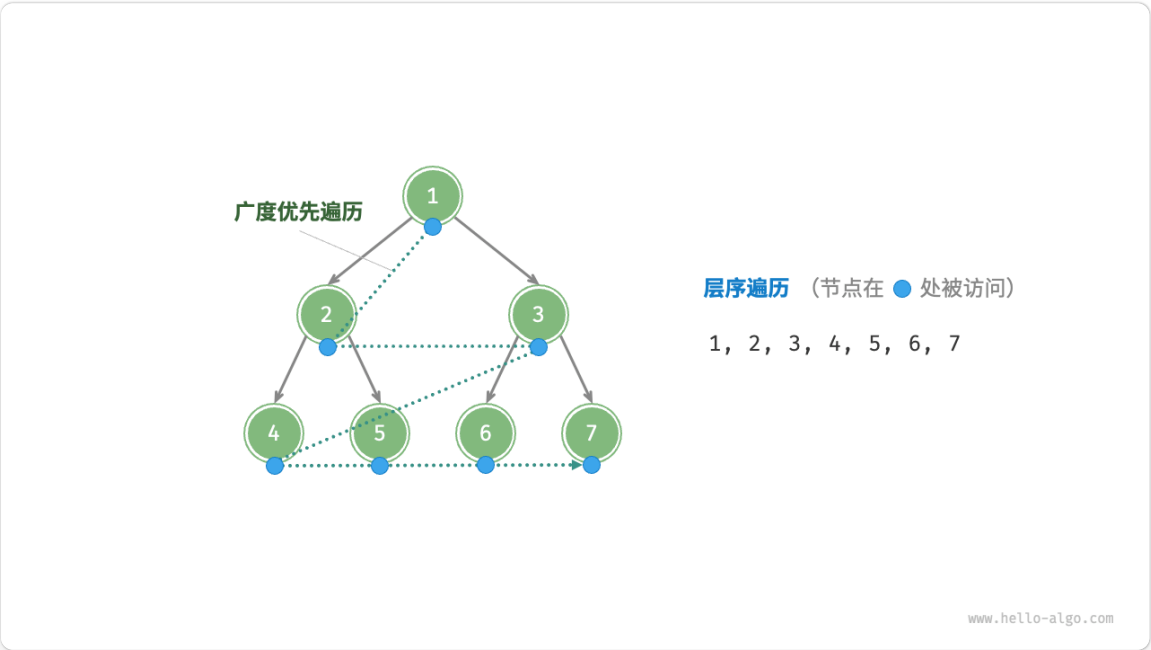


图 7-9 二叉树的层序遍历

#### 1. 代码实现

广度优先遍历通常借助“队列”来实现。队列遵循“先进先出”的规则，而广度优先遍历则遵循“逐层推进”的规则，两者背后的思想是一致的。实现代码如下：

## Python

binary\_tree\_bfs.py

```
def level_order(root: TreeNode | None) -> list[int]:
    """层序遍历"""
    # 初始化队列，加入根节点
    queue: deque[TreeNode] = deque()
    queue.append(root)
    # 初始化一个列表，用于保存遍历序列
    res = []
    while queue:
        node: TreeNode = queue.popleft() # 队列出队
        res.append(node.val) # 保存节点值
        if node.left is not None:
            queue.append(node.left) # 左子节点入队
        if node.right is not None:
            queue.append(node.right) # 右子节点入队
    return res
```

## 2. 复杂度分析

- **时间复杂度为  $O(n)$** ：所有节点被访问一次，使用  $O(n)$  时间，其中  $n$  为节点数量。
- **空间复杂度为  $O(n)$** ：在最差情况下，即满二叉树时，遍历到最底层之前，队列中最多同时存在  $(n + 1)/2$  个节点，占用  $O(n)$  空间。

### 7.2.2 前序、中序、后序遍历

相应地，前序、中序和后序遍历都属于深度优先遍历 (depth-first traversal)，也称深度优先搜索 (depth-first search, DFS)，它体现了一种“先走到尽头，再回溯继续”的遍历方式。

图 7-10 展示了对二叉树进行深度优先遍历的工作原理。**深度优先遍历就像是绕着整棵二叉树的外围“走”一圈**，在每个节点都会遇到三个位置，分别对应前序遍历、中序遍历和后序遍历。

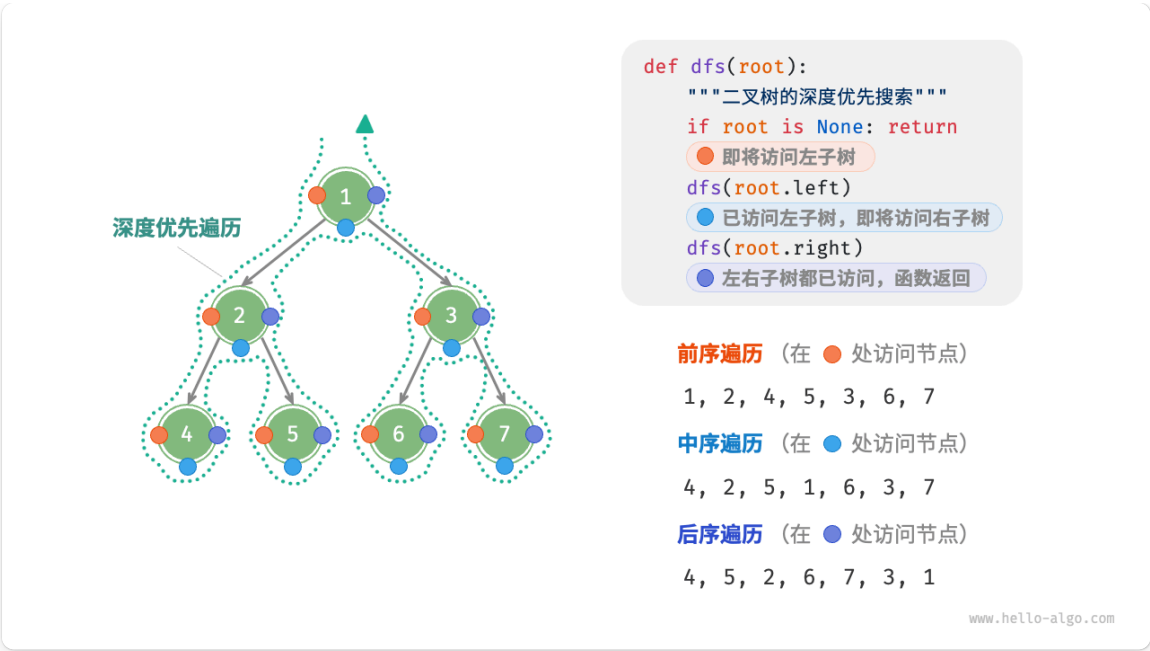


图 7-10 二叉搜索树的前序、中序、后序遍历


1. 代码实现

深度优先搜索通常基于递归实现：

Python

```
binary_tree_dfs.py  
  
def pre_order(root: TreeNode | None):  
    """前序遍历"""  
    if root is None:  
        return  
    # 访问优先级：根节点 -> 左子树 -> 右子树  
    res.append(root.val)  
    pre_order(root=root.left)  
    pre_order(root=root.right)  
  
def in_order(root: TreeNode | None):  
    """中序遍历"""  
    if root is None:  
        return  
    # 访问优先级：左子树 -> 根节点 -> 右子树  
    in_order(root=root.left)  
    res.append(root.val)  
    in_order(root=root.right)  
  
def post_order(root: TreeNode | None):  
    """后序遍历"""  
    if root is None:  
        return
```

```
# 访问优先级：左子树 -> 右子树 -> 根节点
post_order(root=root.left)
post_order(root=root.right)
res.append(root.val)
```

 **Tip**

深度优先搜索也可以基于迭代实现，有兴趣的读者可以自行研究。

图 7-11 展示了前序遍历二叉树的递归过程，其可分为“递”和“归”两个逆向的部分。

- 1. “递”表示开启新方法，程序在此过程中访问下一个节点。
- 2. “归”表示函数返回，代表当前节点已经访问完毕。

<1>

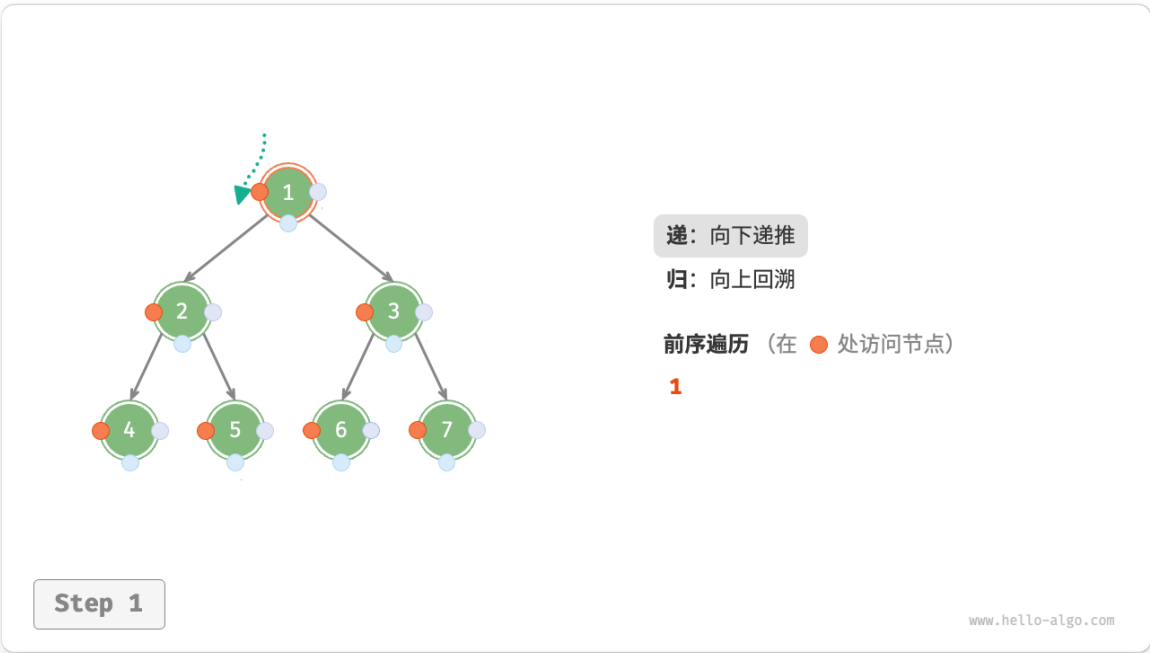


图 7-11 前序遍历的递归过程

2. 复杂度分析

- **时间复杂度为  $O(n)$** ：所有节点被访问一次，使用  $O(n)$  时间。
- **空间复杂度为  $O(n)$** ：在最差情况下，即树退化为链表时，递归深度达到  $n$ ，系统占用  $O(n)$  栈帧空间。