8 第三讲 作业记录（二）

笔记本: 浙江大学《数据结构》
创建时间: 2025/4/5 17:10 **更新时间:** 2025/4/6 16:13
作者: panhengye@163.com
URL: https://pintia.cn/problem-sets/1873565885118418944/exam/problems/type/7?...

03-树2 List Leaves

提交结果

题目	用户	提交时间
03-树2	飞翔的小师弟	2025/04/05 17:04:01
编译器	内存	用时
Python (python3)	3428 / 65536 KB	20 / 400 ms
状态	分数	评测时间
答案正确	25 / 25	2025/04/05 17:04:02

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample 编号乱序，有单边左孩子，中间层少先输出	3428	20	答案正确	13 / 13
1	最大N，有单边右孩子，多层	3256	18	答案正确	5 / 5
2	最小N	3292	17	答案正确	1 / 1
3	每层都有输出，有双孩子	3352	18	答案正确	5 / 5
4	单边树，只有1个输出	3172	17	答案正确	1 / 1

【代码记录】

```
from collections import deque


class TreeNode:
    def __init__(self):
        self.left = None
        self.right = None
        self.value = None


def build_tree(node_count):
    """
    Args:
        node_count: 树的节点数量
    Returns:
        TreeNode: 树的根节点
    Raises:
        ValueError: 当无法确定唯一的根结点时
    """
```

```
    # 存储所有结点
    nodes = {}
    # 存储被引用的结点索引
    referenced_indics = set()


    # 首先创建所有节点
    for i in range(node_count):
        nodes[i] = TreeNode()
        nodes[i].value = i


    # 建立节点之间的连接
    for i in range(node_count):
        left_idx, right_idx = input().strip().split()

        if left_idx != "-":
            left_idx = int(left_idx)
            nodes[i].left = nodes[left_idx]
            referenced_indics.add(left_idx)

        if right_idx != "-":
            right_idx = int(right_idx)
            nodes[i].right = nodes[right_idx]
            referenced_indics.add(right_idx)


    # 查找根结点
    root_index = [i for i in range(node_count) if i not in referenced_indics]
    if len(root_index) != 1:
        raise ValueError("无法确定唯一的根结点")


    return nodes[root_index[0]]


def level_order_traversal(root):
    """
    Args:
        root: 树的根结点
    Returns:
        List: 存放叶子结点值的列表
    """
    if not root:
        return []

    queue = deque([root])
    list_leaves = []


    while queue:
        node = queue.popleft()


        if node.left is None and node.right is None:
            list_leaves.append(str(node.value))


        if node.left is not None:
            queue.append(node.left)
        if node.right is not None:
            queue.append(node.right)

    return list_leaves


def main():
    # 根据输入值"种树"
    nodes = int(input())  # 第一行将输入结点个数
    tree = build_tree(nodes)
```

```
    # 使用层序遍历获取叶子结点的位置
    leaves = level_order_traversal(tree)


    # 结构化输出
    if leaves:
        print(" ".join(leaves))
    else:
        print("-1")

if __name__ == "__main__":
    main()
```

【整体思路】

主流程

- 读取输入的节点数量
- 调用 `build_tree` 构建树
- 调用 `level_order_traversal` 获取叶子节点
- 格式化输出结果

主要功能函数：

- `build_tree(node_count)`：
  - 输入：节点数量
  - 功能：构建二叉树
  - 实现步骤：
    1. 创建所有节点并存储在字典中
    2. 根据输入建立节点之间的连接关系
    3. 通过检查未被引用的节点来确定根节点
  - 输出：返回树的根节点
- `level_order_traversal(root)`：
  - 输入：树的根节点
  - 功能：进行层序遍历，收集叶子节点
  - 实现步骤：
    1. 使用队列进行层序遍历
    2. 检查每个节点是否为叶子节点（左右子节点都为空）
    3. 将叶子节点的值收集到列表中
  - 输出：叶子节点值的列表

【关键点解析1】

问题：build_tree函数中创建结点实例和建立连接两步能否放在一个for循环中处理?

答案：不能。因为如果一边创建一边连接，可能会遇到 KeyError（引用不存在的节点），所以出于数据完整性保证的角度将它们分开是必要的。

【关键点解析2】

用set()类型来查找根结点位置的方法同《第三讲作业记录 （一）》一致，不再赘述

【关键点解析3】

deque（全称double-ended queue）是Python标准库collections模块中的一个类，它提供了一个双端队列的实现。

优势：

- **高效的队列操作**：
  - 在两端添加或删除元素的时间复杂度为O(1)
  - 而普通列表在左端操作的时间复杂度为O(n)
- **线程安全**：
  - deque是线程安全的，可以安全地在多线程环境中使用
- **内存效率**：
  - 比普通列表更节省内存
  - 可以设置最大长度限制

应用举例

```
# 实现一个简单的队列
queue = deque()
queue.append("任务1")    # 入队
task = queue.popleft()  # 出队
```