7 第三讲 作业记录（一）

笔记本: 浙江大学《数据结构》
创建时间: 2025/4/5 12:06 **更新时间:** 2025/4/6 10:10
作者: panhengye@163.com
URL: https://pintia.cn/problem-sets/1873565885118418944/exam/problems/type/7?...

题目: 03-树1 树的同构

提交结果

题目	用户	提交时间
03-树1	飞翔的小师弟	2025/04/05 11:59:37
编译器	内存	用时
Python (python3)	3172 / 65536 KB	16 / 400 ms
状态	分数	评测时间
答案正确	25 / 25	2025/04/05 11:59:38

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample 1 有双边换、单边换，节点编号不同但数据同	3060	15	答案正确	7 / 7
1	sample 2 第3层开始错，每层结点数据对，但父结点不对	3136	16	答案正确	7 / 7
2	结点数不同	3108	16	答案正确	3 / 3
3	空树	3084	15	答案正确	2 / 2
4	只有1个结点，结构同但数据不同	3172	16	答案正确	3 / 3
5	最大N，层序遍历结果相同，但树不同	3156	15	答案正确	3 / 3

【代码记录】

```
class TreeNode:
    """树的节点类"""
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


def build_tree(node_count):
    """
    根据输入构建树结构
    Args:
```

```
        node_count: 树的节点数量
    Returns:
        TreeNode: 树的根节点
    Raises:
        ValueError: 当输入格式不正确或无法构建有效的树时
    """
    if node_count <= 0:
        return None


    # 存储所有节点
    nodes = {}
    # 记录被引用的节点索引
    referenced_indices = set()
    # 存储节点连接信息
    connections = []


    # 读取并验证节点信息
    for i in range(node_count):
        try:
            value, left_idx, right_idx = input().strip().split()
            nodes[i] = TreeNode(value)
            connections.append((left_idx, right_idx))

            # 记录被引用的节点
            if left_idx != "-":
                referenced_indices.add(int(left_idx))
            if right_idx != "-":
                referenced_indices.add(int(right_idx))
        except ValueError:
            raise ValueError(f"第{i+1}行输入格式错误")


    # 建立节点之间的连接
    for i in range(node_count):
        left_idx, right_idx = connections[i]
        if left_idx != "-":
            try:
                nodes[i].left = nodes[int(left_idx)]
            except (ValueError, KeyError):
                raise ValueError(f"无效的左子节点索引: {left_idx}")
        if right_idx != "-":
            try:
                nodes[i].right = nodes[int(right_idx)]
            except (ValueError, KeyError):
                raise ValueError(f"无效的右子节点索引: {right_idx}")


    # 查找根节点（没有被引用的节点）
    root_indices = [i for i in range(node_count) if i not in referenced_indices]
    if len(root_indices) != 1:
        raise ValueError("无法确定唯一的根节点")


    return nodes[root_indices[0]]


def is_isomorphic(tree1, tree2):
    """
    判断两棵树是否同构
    Args:
        tree1: 第一棵树的根节点
        tree2: 第二棵树的根节点
    Returns:
        bool: 如果两棵树同构返回True，否则返回False
    """
    # 两棵树都为空
    if tree1 is None and tree2 is None:
        return True
```

```
    # 只有一棵树为空
    if tree1 is None or tree2 is None:
        return False

    # 根节点值不同
    if tree1.value != tree2.value:
        return False

    # 检查两种情况：
    # 1. 不交换左右子树
    # 2. 交换左右子树
    return (is_isomorphic(tree1.left, tree2.left) and
            is_isomorphic(tree1.right, tree2.right)) or \
           (is_isomorphic(tree1.left, tree2.right) and
            is_isomorphic(tree1.right, tree2.left))


def main():
    """主函数"""
    try:
        # 读取并构建第一棵树
        n1 = int(input())
        tree1 = build_tree(n1)

        # 读取并构建第二棵树
        n2 = int(input())
        tree2 = build_tree(n2)

        # 判断是否同构并输出结果
        result = is_isomorphic(tree1, tree2)
        print("Yes" if result else "No")

    except ValueError as e:
        print(f"输入错误: {e}")
    except Exception as e:
        print(f"程序运行出错: {e}")


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

【整体思路】
和小白专场 - 树的同构 基本一致，详情见课程

①在“栽树”的过程中分别保存左右子结点的信息，通过遍历找出没有被指向的结点的位置，以根结点返回
②在判定是否为同构树的时候要考虑到以下情况

- 都为空 → YES
- 一个为空，另一个不为空 → NO
- 根结点不同 → NO
- 不交换左右子树
- 交换左右子树

【关键点解析1】—— 和教材中有较大差别

```
    # 记录被引用的节点索引
    referenced_indices = set()

    # 查找根节点（没有被引用的节点）
    root_indices = [i for i in range(node_count) if i not in referenced_indices]
    if len(root_indices) != 1:
        raise ValueError("无法确定唯一的根节点")
```

- 这里的处理思路是将索引放入一个set类型中，该数据类型无序且唯一，将其作为一个"检查器"
- 用列表推导式检索没有出现在referenced_indices的索引值，将其以root返回

【关键点解析2】

思考：为什么build_tree函数中没有返回字典nodes，但是不影响后续is_isomorphic中对结点进行访问?

```
def build_tree(node_count):
    nodes = {}  # 临时字典
    # 创建节点实例
    nodes[i] = TreeNode(value)
    # 建立节点之间的引用关系
    nodes[i].left = nodes[left_idx]
    nodes[i].right = nodes[right_idx]
```

- 当我们创建TreeNode类的实例时，每个实例都是一个独立的对象，这些对象通过left和right属性相互引用
- nodes字典只是用来临时存储和访问这些节点实例
- 当函数结束时，字典被释放，但是字典中存储的TreeNode实例并不会被释放
- 因为这些实例之间通过left和right属性相互引用，形成了一个引用链

```
内存中：
TreeNode实例A
  ├── value: "A"
  ├── left: 指向TreeNode实例B
  └── right: 指向TreeNode实例C


TreeNode实例B
  ├── value: "B"
  ├── left: 指向TreeNode实例D
  └── right: None


TreeNode实例C
  ├── value: "C"
  ├── left: None
  └── right: None


TreeNode实例D
  ├── value: "D"
  ├── left: None
  └── right: None
```

总结：

- nodes字典只是一个临时的"容器"
- 真正重要的是TreeNode类的实例以及它们之间的引用关系
- 这些实例和引用关系会一直保存在内存中，直到没有任何变量引用它们时才会被垃圾回收

【关键点解析3】

```
# 存储节点连接信息
connections = []

# 建立节点之间的连接
```

```
    for i in range(node_count):
        left_idx, right_idx = connections[i]
        if left_idx != "-":
            try:
                nodes[i].left = nodes[int(left_idx)]
            except (ValueError, KeyError):
                raise ValueError(f"无效的左子节点索引: {left_idx}")
        if right_idx != "-":
            try:
                nodes[i].right = nodes[int(right_idx)]
            except (ValueError, KeyError):
                raise ValueError(f"无效的右子节点索引: {right_idx}")
```

- 在读取数据的时候将其成对保存到字典中
- 然后分别对左右子结点的值判断，如果不为空就保存"指针"（注意：因为已经在类中做了定义，所以不用再重复将指针指向NULL）

【关键点解析4】

```
nodes[i] = TreeNode(value)
```

由于结点输入的顺序是随机的，所以要有“静态链表”的思想保存每个结点对应的位置，这里可以用 "索引号"："结点" 的字典类型保存

【扩展：set类知识复习】

是一种内置的数据类型，用于表示无序且唯一元素的集合。它具有以下特点：

- 集合中的元素是无序的，无法通过索引来访问元素
- 集合中的元素是唯一的，重复的元素会被自动去除
- 集合是可变的，可以添加、删除元素
    - my_set.add(4) # 添加元素
    - my_set.remove(2) # 删除元素
- 集合中的元素必须是可哈希的，像列表、字典等可变对象不能作为集合的元素，因为它们是不可哈希的
- 集合支持常见的集合运算，如并集、交集、差集
    - union_set = set1.union(set2) # 并集
    - intersection_set = set1.intersection(set2) # 交集
    - difference_set = set1.difference(set2) # 差集
- 可以使用花括号 {} 或者 set() 函数来创建集合。不过，空集合必须使用 set() 来创建