Assignment #5: "树"算：概念、表示、解析、遍历

Updated 2124 GMT+8 March 17, 2024

2024 spring, Complied by 刘子暄 环境科学与工程学院

说明：

1）The complete process to learn DSA from scratch can be broken into 4 parts:

Learn about Time complexities, learn the basics of individual Data Structures, learn the basics of Algorithms, and practice Problems.

2）请把每个题目解题思路（可选），源码Python, 或者C++（已经在Codeforces/Openjudge上AC），截图（包含Accepted），填写到下面作业模版中（推荐使用 typora https://typoraio.cn ，或者用word）。AC 或者没有AC，都请标上每个题目大致花费时间。

3）提交时候先提交pdf文件，再把md或者doc文件上传到右侧“作业评论”。Canvas需要有同学清晰头像、提交文件有pdf、"作业评论"区有上传的md或者doc附件。

4）如果不能在截止前提交作业，请写明原因。

编程环境

（请改为同学的操作系统、编程环境等）

操作系统：Windows 11

Python编程环境: PyCharm Community Edition 2023.3

1. 题目

27638: 求二叉树的高度和叶子数目

http://cs101.openjudge.cn/practice/27638/

思路：整个树的定义思路是节点定义+subtree的形式，整体思路是

1.一个定义两个功能
定义节点
其左节点右节点都定义为None，方便之后赋值为其他节点
高度计算
采用递归函数形式，tree_height这一函数囊括所有子问题，即某节点的高度状态是多少
边界情况：到达叶节点的下一节点，为空，返回-1（因为存在左右节点中有一个为none的情况，所以边界条件是None节点，不是叶节点，否则遍历会有问题）
微操作：返回左右节点中的最大高度，+1（含义是遍历到下一层级）
叶节点计算
递归函数，count_leaf定义类似上方
遍历到None节点时返回0（真正的边界条件）
遍历到叶返回1
微操作：返回左右子树的叶节点的总数的和

2.根据题设的变化
寻找根节点
在录入节点状态的时候，记录录入的左右节点是有父节点的状态（可以使用表加指针），最后查找

代码

```python
class treeNode:
    def __init__(self):
        self.left = None
        self.right = None

def get_height(node):
    if node is None:
        return -1
    return max(get_height(node.left), get_height(node.right)) + 1#这里最后的返回操作必须在上级中，必须

def count_leaf(node):
    if node is None:
        return 0
    if node.left is None and node.right is None:
        return 1
    return count_leaf(node.left) + count_leaf(node.right)

n = int(input())
nodes = [treeNode() for _ in range(n)]
has_fat = [0]*n

for i in range(n):
    left_x, right_x = map(int,input().split())
    if left_x != -1:
        nodes[i].left = nodes[left_x]#应该也搞成节点形式
        has_fat[left_x] = 1#因为value与编号直接相等所以直接换值了，其他情况需要用指针
    if right_x != -1:
        nodes[i].right = nodes[right_x]
        has_fat[right_x] = 1

root_x = has_fat.index(0)
root = nodes[root_x]#两次定位

height = get_height(root)
leaf = count_leaf(root)

print(f"{height} {leaf}")
```

代码运行截图 （至少包含有"Accepted"）

24729: 括号嵌套树

http://cs101.openjudge.cn/practice/24729/

思路：首先，代码中有很多临界条件，要注意随时判断一下，比如stack的空或有，node的none或有

解析树

是用调度场的思路做的，情况分为字母，左括号，右括号

用栈结构来保存拥有子树的节点

字母时，先用node记录当前节点，如果栈不为空，就把这个节点放入父节点（栈顶节点），栈为空，暂时不处理，node就是这个字母

左括号，判断node不为none，则此时的node是某个父节点，压入栈，重置node值

右括号，node值上行变为上一级的节点值

最后的右括号完成之后，node就是root，返回node
遍历
前序遍历
使用递归
output = 目前node的value + 其所有子树的value（链接方式为前序，用递归）
这一步是超级操作
返回
后序遍历
output = 其所有子树的value（链接方式为后序，用递归） + 目前node的value
返回

代码

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []

def parse_tree(nodes):
    stack = []#类似于调度场，把有子树的节点都压入栈，先进后出，保证节点之间的包含关系
    node = None
    for i in nodes:#接下来的操作中，不会出现node没有被录入的情况
        if i.isalpha():
            node = TreeNode(i)
            if stack:
                stack[-1].children.append(node)
        elif i == '(':
            if node:#可能有最外层有括号的格式，所以要判断一下
                stack.append(node)
                node = None
        elif i == ')':
            node = stack.pop()#遇到右括号时，用node接受了目前的“根”节点，有可能更新
    return node

def preorder(node):
    output = [node.value]
    for i1 in node.children:
        output.extend(preorder(i1))
    return ''.join(output)

def postorder(node):
    output1 = []
    for i2 in node.children:
        output1.extend(postorder(i2))
    output1.append(node.value)
    return ''.join(output1)

nodes = input().strip()
nodes = ''.join(nodes.split())#格式化字符串
root = parse_tree(nodes)
if root:
    print(preorder(root))
```

```
print(postorder(root))
```

代码运行截图 （至少包含有"Accepted"）

02775: 文件结构“图”

http://cs101.openjudge.cn/practice/02775/

思路：自己写的输出格式太乱了，参考了夏同学的代码，十分感谢

代码

```
#
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

25140: 根据后序表达式建立队列表达式

http://cs101.openjudge.cn/practice/25140/

思路：建树过程与表达式的转换类似
主要还是对表达式不太熟悉，需要回头理解一下
还有，表达式建树只需要一个表达式，而二叉树遍历表达式必须有中序表达式才能建树

代码

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

def build_tree(s):
    stack = []
    for i in s:
        node = TreeNode(i)
        if i.isupper():
            node.right = stack.pop()
            node.left = stack.pop()
        stack.append(node)
    return stack[0]

def levelorder(root):
    queue = [root]
    travel = []
    while queue:
        node = queue.pop(0)
        travel.append(node.value)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return travel

n = int(input().strip())
for _ in range(n):
    s = input().strip()
    root = build_tree(s)
    queue = levelorder(root)[::-1]
    print(''.join(queue))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

24750: 根据二叉树中后序序列建树

http://cs101.openjudge.cn/practice/24750/

思路：这两个建树题目类似

代码

```python
def build_tree(inorder, postorder):
    if not inorder or not postorder:
        return []

    root_val = postorder[-1]
    root_index = inorder.index(root_val)

    left_inorder = inorder[:root_index]
    right_inorder = inorder[root_index + 1:]

    left_postorder = postorder[:len(left_inorder)]
    right_postorder = postorder[len(left_inorder):-1]

    root = [root_val]
    root.extend(build_tree(left_inorder, left_postorder))
    root.extend(build_tree(right_inorder, right_postorder))

    return root

inorder = input().strip()
postorder = input().strip()
preorder = build_tree(inorder, postorder)
print(''.join(preorder))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

22158: 根据二叉树前中序序列建树

http://cs101.openjudge.cn/practice/22158/

思路：同上，最后一串是为了循环确定终点

代码

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

def build_tree(preorder, inorder):
    if not preorder or not inorder:
        return None
    root_value = preorder[0]
    root = TreeNode(root_value)
    root_index_inorder = inorder.index(root_value)
    root.left = build_tree(preorder[1:1+root_index_inorder], inorder[:root_index_inorder])
    root.right = build_tree(preorder[1+root_index_inorder:], inorder[root_index_inorder+1:])
    return root

def postorder_traversal(root):
    if root is None:
        return ''
    return postorder_traversal(root.left) + postorder_traversal(root.right) + root.value

while True:
    try:
        preorder = input().strip()
        inorder = input().strip()
        root = build_tree(preorder, inorder)
        print(postorder_traversal(root))
    except EOFError:
        break
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

2. 学习总结和收获

如果作业题目简单，有否额外练习题目，比如：OJ“2024spring每日选做”、CF、LeetCode、洛谷等网站题目。

终于好好的把树看懂了，之后再多练一下