Assignment #5: "树"算：概念、表示、解析、遍历

Updated 2124 GMT+8 March 17, 2024

2024 spring, Complied by 胡登科、生科

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：macOS Ventura 13.4.1 (c)

Python编程环境：Spyder IDE 5.2.2, PyCharm 2023.1.4 (Professional Edition)

C/C++编程环境：Mac terminal vi (version 9.0.1424), g++/gcc (Apple clang version 14.0.3, clang-1403.0.22.14.1)

1. 题目

27638: 求二叉树的高度和叶子数目

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

思路：第一个题目比较范式：通过练习再次体会这种链表树的表示方法，越来越觉得树更像是列表映射，而非一棵树了。范式：定义——对于根节点的操作——建立映射关系——从输入中找操作对象——操作——结束。

代码

```
#
class TreeNode:
```

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


def tree_height(node):
    if node is None:
        return -1
    return max(tree_height(node.left), tree_height(node.right)) + 1


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


n = int(input())
nodes = [TreeNode() for _ in range(n)]
has_parent = [False] * n

# 关联列表===建树（做一些判定）
for i in range(n):
    left_index, right_index = map(int, input().split())
    if left_index != -1:
        nodes[i].left = nodes[left_index]
        has_parent[left_index] = True
    if right_index != -1:
        nodes[i].right = nodes[right_index]
        has_parent[right_index] = True

# 我们定义的函数对象最好是根节点，不然会漏掉一部分，所以寻找根节点即没有父节点的节点，查询列表，找到位置，确定root
root_index = has_parent.index(False)
root = nodes[root_index]

height = tree_height(root)
leaves = count_leaves(root)

print(height, leaves)
```

代码运行截图 （至少包含有"Accepted"）

#44389248提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 44389248
题目: 27638
提交人: 2200012286 胡登科
内存: 3668kB
时间: 24ms
语言: Python3
提交时间: 2024-03-24 20:35:55

源代码

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


def tree_height(node):
    if node is None:
        return -1
    return max(tree_height(node.left), tree_height(node.right)) + 1


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


n = int(input())
nodes = [TreeNode() for _ in range(n)]
has_parent = [False] * n

# 关联列表===建树（做一些判定）
for i in range(n):
    left_index, right_index = map(int, input().split())
    if left_index != -1:
        nodes[i].left = nodes[left_index]
        has_parent[left_index] = True
    if right_index != -1:
        nodes[i].right = nodes[right_index]
        has_parent[right_index] = True

# 我们定义的函数对象最好是根节点，不然会漏掉一部分，所以寻找根节点即没有父节点的节点，
root_index = has_parent.index(False)
root = nodes[root_index]

height = tree_height(root)
leaves = count_leaves(root)
```

24729: 括号嵌套树

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

思路：先建树——写操作——输出

这个题目难在一多子树，二如何将树后续输出。输出树的操作本质在于遍历树。我们需要按照输出格式的要求去遍历树，例如后续遍历，就只能先递归遍历找到最下面的节点，然后在归的规程中整理表达式。而前缀输出在递的时候（for 循环就能解决，不需要递归）就得去整理输出了，反而不需要太多的返回值。这个题目是比较好的递归题目。

代码

```
#
# 括号嵌套 多叉树
# 思路：先建树，后输出

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

```
# 建树
def parse_tree(s):
    stack = []
    node = None
    for char in s:
        if char.isalpha():
            node = TreeNode(char)
            if stack:
                stack[-1].children.append(node)
        elif char == '(':
            if node:
                stack.append(node)
                node = None
        elif char == ')':
            if stack:
                node = stack.pop()
    return node


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


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


def main():
    s = input().strip()
    s = ''.join(s.split())
    root = parse_tree(s)
    if root:
        print(preorder(root))  # 输出前序遍历序列
        print(postorder(root))  # 输出后序遍历序列
    else:
        print("input tree string error!")


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

代码运行截图 （至少包含有"Accepted"）

#44390422提交状态

状态: Accepted

源代码

```
# 括号嵌套 多叉树
# 思路：先建树，后输出

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


# 建树
def parse_tree(s):
    stack = []
    node = None
    for char in s:
        if char.isalpha():
            node = TreeNode(char)
            if stack:
                stack[-1].children.append(node)
        elif char == '(':
            if node:
                stack.append(node)
                node = None
        elif char == ')':
            if stack:
                node = stack.pop()
    return node


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


def postorder(node):
    output = []
    for child in node.children:
        output.extend(postorder(child))
    output.append(node.value)
    return ''.join(output)
```

02775: 文件结构“图”

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

思路：见程序。感觉用这个建树的方法好麻烦，而且还很难的做到对文件排序。道阻且长啊。打了三个小时还不是很对，哭了。感觉对于树的了解加深了，但是自信心受挫严重。

代码

```
#
# 文件结构图
# 思路 先建树，后输出。 建树思路：读取file时，判断栈是否为空。若空则进root文件，不为空就append。读取dir
# 若栈为空，进栈，若不为空，压入前dir（用列表），后进栈。读取 ] 若栈空，报错，若不空，将栅尾dir弹出。读取*
# 结束该次输入。读取#，结束程序。

class DirStruc:
    def __init__(self, loc, name):
        self.loc = loc
        self.file = []
        self.name = name


# 建树
def build_struc():
    file_type = DirStruc(0, ' ')  # 初始化输入类型
    root = DirStruc(0, 'root')  # 初始化文件根
    dir_stack = []
    dir_stack.append(root)  # 将根文件入栈
    root.loc = len(dir_stack) - 1  # 将根的位置定位0
    while True:
        file_type = DirStruc(0, ' ')  # 初始化输入类型
        a = str(input())
        if a == '':
            return root, 2
        if a == '*':
            death = 0
            break
        file_type.name = a  # 录入文件
        if a == '#':
            death = 1
            break

        if file_type.name[0] == 'f':  # f
            if dir_stack[-1].name[0] == 'd':  # 栈中有d
                dir_stack[-1].file.append(file_type)  # f->d
                file_type.loc = dir_stack[-1].loc  # f_loc = d_loc

            else:
                root.file.append(file_type)  # 为根文件
                file_type.loc = dir_stack[-1].loc  # loc

        elif file_type.name[0] == 'd':  # d

            file_type.loc = dir_stack[-1].loc + 1  # loc+1
            dir_stack[-1].file.append(file_type)
            dir_stack.append(file_type)  # 入栈

        elif file_type.name[0] == ']':
            if dir_stack[-1].name[0] == 'd':
                dir_stack.pop()

            else:
```

```
                print("Error")
    return root, death


# 输出树
# 思路，通过返回root进行输出，判断loc，和.file是否为空
def print_struc(root):
    for item in root.file:
        if item.loc != 0:
            print("|     " * int(item.loc) + item.name)
            print_struc(item)
    if len(root.file) == 0:
        return
    num = 0
    for item in root.file:
        if item.loc == 0:
            num += 1
    for _ in range(1, num + 1):
        print(f"file{_}")


n = 0
while True:
    n += 1
    root, death = build_struc()
    if death == 1:
        break
    elif death == 2:
        print(f"DATA SET {n}:")
        print("ROOT")
    else:
        print('\n'f"DATA SET {n}:")
        print("ROOT")
        print_struc(root)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

```
```

#44394095提交状态

状态: Accepted

源代码

25140: 根据后序表达式建立队列表达式

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

思路：先读懂题目要求，将后续表达式建树，通过栈的方式。然后递归将元素正向遍历，然后反向输出。

代码

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


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


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


n = int(input().strip())
for _ in range(n):
    postfix = input().strip()
    root = build_tree(postfix)
    queue_expression = level_order_traversal(root)[::-1]
    print(''.join(queue_expression))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

源代码

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


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


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


n = int(input().strip())
for _ in range(n):
    postfix = input().strip()
    root = build_tree(postfix)
    queue_expression = level_order_traversal(root)[::-1]
    print(''.join(queue_expression))
```

24750: 根据二叉树中后序序列建树

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

思路： 后序遍历提供一个节点信息，而中序遍历可以左右分割左右子树。本题的特点在于如何利用已知信息去建一棵树。这种建树的思路与链表树的类型不太一样，链表树偏向于解决多叉树，大量输入的问题。而表达式树偏向于解决有限信息建树的。前者通过if判断节点走向，后者通过一些规律，例如后序的尾巴上顶数，中序分左右二子树的规律。

代码

```
# 根据二叉树中后序序列建树
```

```python
# 思路：后序遍历提供一个节点信息，而中序遍历可以左右分割左右子树

def build_tree(inorder, postorder):
    if not inorder or not postorder:  # 如果有一个为空就返回空列表
        return []

    root_val = postorder[-1]  # 后序确定根值
    root_index = inorder.index(root_val)  # 中序确定根植的位置

    left_inorder = inorder[:root_index]  # 区分左右子树
    right_inorder = inorder[root_index + 1:]

    left_postorder = postorder[:len(left_inorder)]  # 根据后序遍历的特点，创建新的左右子树的后序遍历，方便找节点
    right_postorder = postorder[len(left_inorder):-1]

    root = [root_val]
    root.extend(build_tree(left_inorder, left_postorder))  # 开始递归
    root.extend(build_tree(right_inorder, right_postorder))

    return root


def main():
    inorder = input().strip()
    postorder = input().strip()
    preorder = build_tree(inorder, postorder)
    print(''.join(preorder))


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

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

源代码

```
"""
后序遍历的最后一个元素是树的根节点。然后，在中序遍历序列中，根节点将左右子树分开。
可以通过这种方法找到左右子树的中序遍历序列。然后，使用递归地处理左右子树来构建整个树。
"""

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


def main():
    inorder = input().strip()
    postorder = input().strip()
    preorder = build_tree(inorder, postorder)
    print(''.join(preorder))


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

22158: 根据二叉树前中序序列建树

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

思路：与上图类似，甚至更简单。

代码

```
#
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"）

状态：Accepted

源代码

```
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.right = build_tree(preorder[1 + root_index_inorder:], inorder[
    return root


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


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

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。

本周练习量少，非常惭愧，感觉自己每天的时间不是很够用。一开始打代码就容易着魔，变成魔怔人。经常会有思路清晰但是代码中有很多bug的情况。就是今天打的那个文件图，思路非常简单，就是一个if建链表树，但是中途过程思考少了，考虑不周全，会漏掉一些特殊情况。实话实说这个题做出来巨有成就感，就是感觉第一次自己建出来一颗树，熟练运用递归，熟练分析问题。

最近的日子也是越来越忙，希望身体健康，每天投入时间逐渐恢复，提升效率。本周基本上没有做其他的练习，专注学习树的程序就已经很艰难了。