Assignment #5: "树"算：概念、表示、解析、遍历

Updated 2124 GMT+8 March 17, 2024

2024 spring, Complied by 夏天，生命科学学院

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 10 家庭版

Python编程环境：Spyder python(3.11)

1. 题目

27638: 求二叉树的高度和叶子数目

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

思路：题目中似乎没有说0不一定是根节点（样例数据也说明了这一点）于是直接遍历每一个节点，求节点的子树的高度，再找出最大值即可；没有孩子节点的节点树就是叶子数。

代码

```
class Treenode:
    def __init__(self,x):
        self.val=x
        self.left=None
        self.right=None
def build_tree(n,nodes):
    dic={i:Treenode(i)for i in range(n)}
    for i,(left,right) in enumerate(nodes):
        if left!=-1:
            dic[i].left=dic[left]
        if right!=-1:
            dic[i].right=dic[right]
    return dic
def max_height(root):
    if root is None:
        return -1
    else:
        left_height=max_height(root.left)
        right_height=max_height(root.right)
        return max(left_height,right_height)+1
n=int(input())
nodes=[]
count=0
for _ in range(n):
    left,right=map(int,input().split())
    nodes.append((left,right))
    if left==right==-1:
        count+=1
dic=build_tree(n,nodes)
height=0
for i in range(n):
    root=dic[i]
    height=max(max_height(root),height)
print(height,count)
```

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：见注释

代码

```
class Treenode:
    def __init__(self,value):
        self.value=value
        self.children=[]
def parse_tree(s):
    stack=[] #储存双亲节点
    for char in s:
        if char.isalpha(): #是字母，创建节点
            node=Treenode(char)
            if stack: #栈非空，则当前节点一定是栈顶节点的孩子节点
                stack[-1].children.append(node)
        if char=='(': #是(，则当前节点可能是双亲节点
            stack.append(node)
        if char==')': #是),则栈顶节点的孩子节点已全部列出
            node=stack.pop()
    return node #返回根节点
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)  #所有孩子节点输出之后在输出双亲节点
    return ''.join(output)
s=input()
root=parse_tree(s)
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：见注释

代码

```
class root:
    def __init__(self,name):
        self.name=name
        self.dir=[]
        self.file=[]
    def build(self,s):
        if s[0]=='f': #如果root下是文件，则加入root.file中
            self.file.append(s)
        elif s[0]=='d': #如果root下是目录，则加入root.dir中，同时把目录作为新的root
            dir_=root(s)
            self.dir.append(dir_)
            while True: #在新root下重复上述过程，遇到]即停止该轮操作
                s=input()
                if s==']':
                    break
                dir_.build(s)
def graph(r,i=0): #生成“图”，r表示当前目录/根，i表示层数
    print('|     '*i+r.name)
    if r.dir: #如果有子目录，则进入下一层，重复上述操作
        for a in r.dir:
            graph(a,i+1)
    r.file.sort() #按字母表顺序排好序
    for b in r.file:
        print('|     '*i+b)
j=0 #记录测试数据的编号
while True:
    s=input()
    if s=='#': #结束接收数据
        break
    j+=1
    r=root('ROOT') #初始的ROOT
    while True:
        r.build(s) #生成“树”
        s=input()
        if s=='*': #一组测试数据结束
            break
    print(f'DATA SET {j}:')
    graph(r,0)
    print()
```

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：跟着提示走即可，用后序表达式建立表达式树，再按层次遍历，将遍历结果前后颠倒得到答案

代码

```
from collections import deque
class Treenode:
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None
def build_tree(postfix):
    stack=[]
    for char in postfix:
        if char.islower():
            node=Treenode(char)
            stack.append(node)
        else:
            right=stack.pop()
            left=stack.pop()
            new_node=Treenode(char)
            new_node.left=left
            new_node.right=right
            stack.append(new_node)
    return stack[0]
def level_traversal(root):
    result=[]
    if not root:
        return result
    queue=deque([root])
    while queue:
        node=queue.popleft()
        result.append(node.value)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return result[::-1]
n=int(input())
for _ in range(n):
    postfix=input()
    root=build_tree(postfix)
    result=level_traversal(root)
    print(''.join(result))
```

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：见注释

代码

```
def build_tree(inorder,postorder):
    if not inorder or not postorder:
        return []
    root=postorder[-1] #后序表达式的最后一个一定为根节点
    root_index=inorder.index(root)
    left_inorder=inorder[:root_index] #左子树的中序表达式
    right_inorder=inorder[root_index+1:] #右子树的中序表达式
    left_postorder=postorder[:len(left_inorder)] #左子树的后序表达式
    right_postorder=postorder[len(left_inorder):-1] #右子树的后序表达式
    tree=[root]
    tree.extend(build_tree(left_inorder,left_postorder)) #将左子树视为一个新的二叉树，递归
    tree.extend(build_tree(right_inorder,right_postorder)) #将右子树视为一个新的二叉树，递归
    #注意：根据前序遍历的要求，左子树先右子树后
    return tree
inorder=input()
postorder=input()
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：和上一道题基本一样，注意后序表达式需要先创建空列表，将孩子节点全部加入列表之后再将双亲节点加入列表

代码

```
    def build_tree(preorder,inorder):
        if not preorder or not inorder:
            return []
        root=preorder[0]
        root_index=inorder.index(root)
        left_inorder=inorder[:root_index]
        right_inorder=inorder[root_index+1:]
        left_preorder=preorder[1:len(left_inorder)+1]
        right_preorder=preorder[len(left_inorder)+1:]
        tree=[]
        tree.extend(build_tree(left_preorder,left_inorder))
        tree.extend(build_tree(right_preorder,right_inorder))
        tree.append(root)
        return tree
    while True:
        try:
            preorder=input()
            inorder=input()
            print(''.join(build_tree(preorder,inorder)))
        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。

上周初次接触树这个数据结构之后，通过一些渠道（CSDN博客，知乎，菜鸟教程等）及本周课上的讲解学习了相关概念。这里想简单地写一下自己的理解，如果有不正确的地方还望老师指出！

①用class建立树的过程是比较直观的：如果根节点已知，那么建树的过程就很像一棵树自然生长的过程，生根之后不断分叉；如果根节点不已知，那么建树的过程就很像嫁接——有很多已知的枝条，然后根据信息不断把这些枝条接到主干上。

②高度V.S.深度：最长路径的结点数（树的深度）减1为树的高度。为了便于区分，可以联系生活实际，通常我们说树的高度一般都指它的地上部分，从这个角度思考的话计算高度要减1（根节点）就很自然了。

③节点：没有双亲节点的节点就是根节点，没有孩子节点的节点就是叶子，27638：求二叉树的高度和叶子数目同时考察了这两个概念（不过做作业时还不知道怎么找根节点，于是直接遍历了所有节点，之后会尝试写一下寻找根节点的代码）

④四种遍历：前序、中序、后序、层次。其中前三个都用到了递归的思想，即将左右子树作为一棵新的树进行遍历，代码也很相似，唯一不同点就是什么时候将双亲节点加入列表（与名字相对应）；24750：根据二叉树中后序序列建树和22158：根据二叉树前中序序列建树考察了这三种遍历的理解，注意到前序遍历的第一个一定是根节点，后序遍历的最后一个一定是根节点，找到根节点后借助中序表达式可以得到左右子树，然后递归即可（另外做这两道题的时候还思考了一个问题：根据二叉树前后序序列能否得到中序表达式？答案是否定的，因为仅能得到根节点，得不到子树的详细信息） 而按层次遍历似乎是我们更习惯的一种遍历方式，从上到下，从左到右，代码需要用到队列（25140：根据后序表达式建立队列表达式）

⑤解析树：感觉有点像④的逆过程，与波兰表达式、逆波兰表达式有很密切的关系，代码需要用到栈，由此可以看出栈和队列这两个数据结构很重要

⑥后记：现在处于一种能看懂代码但是自己完整写出来还有点困难的状态（如02775：文件结构“图”），可能多刷题能解决这个问题。不过做作业时也发现这部分的代码基本上都是固定模板，直接加入cheat sheet！