Assignment #5: "树"算：概念、表示、解析、遍历

2024 spring, Complied by 胡豪俊 工学院

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

操作系统：Window11

Python编程环境：Visiual Studio Code

1. 题目

27638: 求二叉树的高度和叶子数目

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

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)##不用+1，算的是子节点，用不着根结点

n=int(input())
nodes=[TreeNode() for _ in range(n)]
has_parent=[False]*n
for i in range(n):
    left_index,right_index=map(int,input().split())
    if left_index!=-1:
        nodes[i].left=nodes[left_index]##这里不用-1是因为输入的自带是从0开始的编号
        has_parent[left_index]=True

    if right_index!=-1:
        nodes[i].right=nodes[right_index]
        has_parent[right_index]=True

root_index=has_parent.index(False)
root=nodes[root_index]
height=tree_height(root)
leaves=count_leaves(root)
print(f"{height} {leaves}")
##其实就是用递归来实现的，重点是要找到根结点，其余我觉得和前面的求二叉树深度是一样的，当然需要注意一下高度和深度的定义
```

#44399679提交状态

查看 提交 统计 提问

状态: Accepted

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)##不用+1，算的

n=int(input())
nodes=[TreeNode() for _ in range(n)]
has_parent=[False]*n
for i in range(n):
    left_index,right_index=map(int,input().split())
    if left_index!=-1:
        nodes[i].left=nodes[left_index]##这里不用-1是因为输入的自带是从0开始
        has_parent[left_index]=True

    if right_index!=-1:
        nodes[i].right=nodes[right_index]
        has_parent[right_index]=True

root_index=has_parent.index(False)
root=nodes[root_index]
height=tree_height(root)
leaves=count_leaves(root)
print(f"{height} {leaves}")
```

基本信息

#:	44399679
题目:	27638
提交人:	22n2200011800
内存:	3720kB
时间:	26ms
语言:	Python3
提交时间:	2024-03-25 20:02:43

24729: 括号嵌套树

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

代码

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

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))
        print(postorder(root))
    else:
        print("input tree string error!")

if __name__=="__main__":
    main()
##一开始往引号下意识里打了空格错了几回...这题我觉得和前面的前序表达式和后续表达式他们的转换是非常像的，引入一个栈，重点考察括号的行为，括号会提醒我们什么时候出现新的分支，由此便能解决问题
```

状态: Accepted

源代码

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

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))
        print(postorder(root))
    else:
        print("input tree string error!")

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

基本信息

#:	44400123
题目:	24729
提交人:	22n2200011800
内存:	3668kB
时间:	22ms
语言:	Python3
提交时间:	2024-03-25 20:32:44

02775: 文件结构“图”

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

代码

```
class Node:
    def __init__(self,name):
        self.name=name
        self.dirs=[]
        self.files=[]

def print_structure(node,indent=0):
    prefix="|     "*indent
    print(prefix+node.name)
    for dir in node.dirs:
        print_structure(dir,indent+1)
    for file in sorted(node.files):
        print(prefix+file)
dataset=1
```

```
datas=[]
temp=[]
while True:
    line=input()
    if line=="#":
        break
    if line=="*":
        datas.append(temp)
        temp=[]
    else:
        temp.append(line)

for data in datas:
    print(f"DATA SET {dataset}:")
    root=Node("ROOT")
    stack=[root]
    for line in data:
        if line[0]=="d":
            dir=Node(line)
            stack[-1].dirs.append(dir)
            stack.append(dir)
        elif line[0]=="f":
            stack[-1].files.append(line)
        elif line=="]":
            stack.pop()
    print_structure(root)
    if dataset<len(datas):
        print()
    dataset+=1
```

##挺复杂的...虽然从直觉上，这是一件非常显然的事情，看一眼我们就能知道文件的分布结构，但实际上真正要用计算机的语言去实现需要注意很多细节，参考了题解，自己写估计写不太出来。

#44400937提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class Node:
    def __init__(self,name):
        self.name=name
        self.dirs=[]
        self.files=[]

def print_structure(node,indent=0):
    prefix='|     '*indent
    print(prefix+node.name)
    for dir in node.dirs:
        print_structure(dir,indent+1)
    for file in sorted(node.files):
        print(prefix+file)
dataset=1
datas=[]
temp=[]
while True:
    line=input()
    if line=="#":
        break
    if line=="*":
        datas.append(temp)
        temp=[]
    else:
        temp.append(line)

for data in datas:
    print(f"DATA SET {dataset}:")
    root=Node("ROOT")
    stack=[root]
    for line in data:
        if line[0]=="d":
            dir=Node(line)
            stack[-1].dirs.append(dir)
            stack.append(dir)
        elif line[0]=="f":
            stack[-1].files.append(line)
        elif line=="]":
            stack.pop()
    print_structure(root)
    if dataset<len(datas):
        print()
    dataset+=1
```

基本信息

#: 44400937
题目: 02775
提交人: 22n2200011800
内存: 3668kB
时间: 25ms
语言: Python3
提交时间: 2024-03-25 21:21:49

English 帮助 关于

25140: 根据后序表达式建立队列表达式

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

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 i in range(n):
    postfix=input().strip()
    root=build_tree(postfix)
    queue_expression=level_order_traversal(root)[::-1]
    print("".join(queue_expression))
```

##思路里很自然地需要引入栈（不得不感叹前面一节栈的学习还是非常有用的，尤其是前序表达式后序表达式互相转换的部分），最后输出的[::-1]这里学题解的，这些细节熟练起来其实能剩下很多时间

#44402334提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 44402334
题目: 25140
提交人: 22n2200011800
内存: 3644kB
时间: 28ms
语言: Python3
提交时间: 2024-03-25 23:05:26

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 i in range(n):
    postfix=input().strip()
    root=build_tree(postfix)
    queue_expression=level_order_traversal(root)[::-1]
    print("".join(queue_expression))
```


English 帮助 关于

24750: 根据二叉树中后序序列建树

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

代码

```
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()
```

##这题感觉和后面那题很像，当然也有所不同。本质上，单个序列，其实是无法确定唯一一个树的，结点随便怎么分都可以，但是一旦拥有两个表达式，例如此处的中序和后序，那么这个树便被唯一确定了，利用这个想法以及递归的思路提取中序和后序的信息，从而建树

#44402522提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
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()
```

基本信息

#: 44402522
题目: 24750
提交人: 22n2200011800
内存: 3668kB
时间: 23ms
语言: Python3
提交时间: 2024-03-25 23:24:33

English 帮助 关于

22158: 根据二叉树前中序序列建树

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

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_inorde
r])

 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
```

##和前面那个是类似的，同样是提取信息，然后建树。要注意的是输入的情况要用try except判断，最开始直接在原始代码上修改错了好几回，最后才发现

#44402714提交状态

查看 提交 统计 提问

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_
    root.right=build_tree(preorder[1+root_index_inorder:],inorder[root_
    return root
def postorder_traversal(root):
    if root is None:
        return ""
    return postorder_traversal(root.left)+postorder_traversal(root.right
while True:
    try:
        preorder=input().strip()
        inorder=input().strip()
        root=build_tree(preorder,inorder)
        print(postorder_traversal(root))
    except EOFError:
        break
```

基本信息

#:	44402714
题目:	22158
提交人:	22n2200011800
内存:	3524kB
时间:	24ms
语言:	Python3
提交时间:	2024-03-25 23:49:50

2. 学习总结和收获

还是有点难的，主要体现在实现上，对于思路其实大部分问题都能想明白，但具体到细节上我就会出很多稀奇古怪的错误，比如那道文件结构图。而且这次作业比较尴尬的是，我先看了老师发在群里的讲义，没看作业是啥之前自己做了一遍里面的题目，开始写作业的时候发现大部分题目参考题解都被我复现了，所以挺多直接用了第一次写的代码。当然我也看了群里老师发的其他同学的题解，学到了很多。对树这块目前的掌握情况远远不够，要继续加大投入