Assignment #5: "树"算：概念、表示、解析、遍历

Updated 2124 GMT+8 March 17, 2024

2024 spring, Complied by 武昱达 23工院

编程环境

操作系统：Windows 11

Python编程环境：PyCharm 2023.1.4 (Professional Edition)

1. 题目

27638: 求二叉树的高度和叶子数目

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

思路：

代码

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

n=int(input())
nodes=[TreeNode() for _ in range(n)]
root=TreeNode()

for i in range(n):
    nodes[i].index=i

for node_index in range(n):
    left,right=map(int,input().split())
    if left!=-1:
        nodes[left].parent=nodes[node_index]
        nodes[node_index].left=nodes[left]
    if right!=-1:
        nodes[right].parent=nodes[node_index]
        nodes[node_index].right=nodes[right]

leaves=0
for node in nodes:
    if node.parent==None:
        root=node
    if node.left==None and node.right==None:
        leaves+=1

```

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

print(TreeHeight(root),leaves)
```

代码运行截图 （至少包含有"Accepted"）

#43933259提交状态

查看 提交 统计 提问

状态: Accepted

源代码

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

n=int(input())
nodes=[TreeNode() for _ in range(n)]
root=TreeNode()

for i in range(n):
    nodes[i].index=i

for node_index in range(n):
    left,right=map(int,input().split())
    if left!=-1:
        nodes[left].parent=nodes[node_index]
        nodes[node_index].left=nodes[left]
    if right!=-1:
        nodes[right].parent=nodes[node_index]
        nodes[node_index].right=nodes[right]

leaves=0
for node in nodes:
    if node.parent==None:
        root=node
    if node.left==None and node.right==None:
        leaves+=1

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

print(TreeHeight(root),leaves)
```

基本信息

#: 43933259
题目: 27638
提交人: 23n2300011119 (武)
内存: 3656kB
时间: 20ms
语言: Python3
提交时间: 2024-02-19 21:40:53

©2002-2022 POJ 京ICP备20010980号-1 English 帮助 关于

24729: 括号嵌套树

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

思路:

代码

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

non_l=('(',')',',')
def BuildTree(s:str):
    # 该函数的作用是：通过s建树存到内存空间中，并返回根节点
    '''
    实现原理：
    发现字母即加入父节点的子列表中，子列表的正序即树从左到右；
    一个node循环指向某个实例，在遇见左括号时让这个实例入栈，
    并且在下一次最先弹出并且被当做父节点。
```

```
    遇见右括号则把栈顶元素弹出，意味着该节点的子树构建完成。
    '''
    stack=[]
    node=None
    for char in s:
        if char.isalpha():
            node=TreeNode(char)
            if stack:
                # 如果栈不为空，把节点作为子节点加入栈顶节点的子节点列表中
                stack[-1].children.append(node)
        elif char=='(':
            if node:
                stack.append(node)
                node=None
        elif char==')':
            if stack:
                node=stack.pop()
    return node

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

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

def main():
    s = input().strip()
    s = ''.join(s.split())  # 去掉所有空白字符
    root = BuildTree(s)  # 解析整棵树
    if root:
        print(pre_order(root))  # 输出前序遍历序列
        print(post_order(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"）

4196561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.children=[]

non_l=('(',')',',')
def BuildTree(s:str):
    # 该函数的作用是：通过s建树存到内存空间中，并返回根节点
    '''
    实现原理：
    发现字母即加入父节点的子列表中，子列表的正序即树从左到右；
    一个node循环指向某个实例，在遇见左括号时让这个实例入栈，
    并且在下一次最先弹出并且被当做父节点，
    遇见右括号则把栈顶元素弹出，意味着该节点的子树构建完成。
    '''
    stack=[]
    node=None
    for char in s:
        if char.isalpha():
            node=TreeNode(char)
            if stack:
                # 如果栈不为空，把节点作为子节点加入栈顶节点的子节点列表中
                stack[-1].children.append(node)
        elif char=='(':
            if node:
                stack.append(node)
                node=None
        elif char==')':
            if stack:
                node=stack.pop()
    return node

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

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

def main():
    s = input().strip()
    s = ''.join(s.split())  # 去掉所有空白字符
    root = BuildTree(s)  # 解析整棵树
    if root:
        print(pre_order(root))  # 输出前序遍历序列
        print(post_order(root))  # 输出后序遍历序列
    else:
        print("input tree string error!")

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

基本信息

#: 44196561
题目: 24729
提交人: 23n2300011119 (武)
内存: 3672kB
时间: 30ms
语言: Python3
提交时间: 2024-03-13 15:23:07

English 帮助 关于

02775: 文件结构“图”

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

思路:

代码

```
class TreeNode:
    def __init__(self,val):
        self.val = val
        self.files = []
        self.dirs = []
        self.parent=None

def GraphTree(l:list):
```

```
    root=TreeNode('ROOT')
    current_node=root
    for name in l:
        if name[0]=='f':
            tmp=TreeNode(name)
            tmp.parent=current_node
            current_node.files.append(tmp)
        elif name[0]=='d':
            tmp=TreeNode(name)
            tmp.parent=current_node
            current_node.dirs.append(tmp)
            current_node=tmp
        else:
            # name==']':
            current_node=current_node.parent
    return root

res=['ROOT']
def DrawGraph(root:TreeNode,depth):
    # 函数的功能是把该根节点目录下的所有打印行添加到res中
    global res
    for dir in root.dirs:
        # 对于目录中的目录，同样执行该操作
        res.append("|     "*(depth+1)+dir.val)
        DrawGraph(dir,depth+1)

    root.files.sort(key=lambda x:x.val)
    for file in root.files:
        # 对于目录中的文件执行操作
        res.append('|     '*depth+file.val)
    return

stack=[[]]
while (n:=input())!='#':
    if n!='*':stack[-1].append(n)
    else:stack.append([])
stack.pop()

for i in range(len(stack)):
    l,res=stack[i],['ROOT']
    root=GraphTree(l)
    DrawGraph(root,0)
    print('DATA SET {}:'.format(i+1))
    for j in res:
        print(j)
    if i!=len(stack)-1:
        print(" ")
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44278732提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class TreeNode:
    def __init__(self,val):
        self.val = val
        self.files = []
        self.dirs = []
        self.parent=None

def GraphTree(l:list):
    root=TreeNode('ROOT')
    current_node=root
    for name in l:
        if name[0]=='f':
            tmp=TreeNode(name)
            tmp.parent=current_node
            current_node.files.append(tmp)
        elif name[0]=='d':
            tmp=TreeNode(name)
            tmp.parent=current_node
            current_node.dirs.append(tmp)
            current_node=tmp
        else:
            # name==']':
            current_node=current_node.parent
    return root

res=['ROOT']
def DrawGraph(root:TreeNode,depth):
    # 函数的功能是把该根节点目录下的所有打印行添加到res中
    global res
    for dir in root.dirs:
        # 对于目录中的目录，同样执行该操作
        res.append('|     '*(depth+1)+dir.val)
        DrawGraph(dir,depth+1)

    root.files.sort(key=lambda x:x.val)
    for file in root.files:
        # 对于目录中的文件执行操作
        res.append('|     '*depth+file.val)
    return

stack=[[]]
while (n:=input())!='#':
    if n!='*':stack[-1].append(n)
    else:stack.append([])
stack.pop()

for i in range(len(stack)):
    l,res=stack[i],['ROOT']
    root=GraphTree(l)
    DrawGraph(root,0)
    print('DATA SET {}:'.format(i+1))
    for j in res:
        print(j)
    if i!=len(stack)-1:
        print("")
```

基本信息

#: 44278732
题目: 02775
提交人: 23n2300011119 (武)
内存: 3624kB
时间: 23ms
语言: Python3
提交时间: 2024-03-17 20:59:19

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/

思路:

代码

```
from collections import deque

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

def Parse_tree(s:deque):
    l=len(s)
```

```
    stack = []
    node_dict=dict()
    while s:
        name=s.popleft()
        node_dict[name]=TreeNode(name)
        if name.islower():
            stack.append(node_dict[name])
        else:
            num_2=stack.pop()
            num_1=stack.pop()
            node_dict[name].left=num_1
            node_dict[name].right=num_2
            stack.append(node_dict[name])
    root=stack[0]
    res=Tree_BFS(root,l)
    return res

# 用BFS的方法遍历二叉树
def Tree_BFS(root,l):
    queue=deque()
    queue.append(root)
    res=[root.value]
    while len(res)<l:
        a=queue.popleft()
        if a.left!=None:
            res.append(a.left.value)
            queue.append(a.left)
        if a.right!=None:
            res.append(a.right.value)
            queue.append(a.right)
    return res

for _ in range(int(input())):
    raw=deque(input())
    print("".join(reversed(Parse_tree(raw))))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44282867提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
from collections import deque

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

def Parse_tree(s:deque):
    l=len(s)
    stack = []
    node_dict=dict()
    while s:
        name=s.popleft()
        node_dict[name]=TreeNode(name)
        if name.islower():
            stack.append(node_dict[name])
        else:
            num_2=stack.pop()
            num_1=stack.pop()
            node_dict[name].left=num_1
            node_dict[name].right=num_2
            stack.append(node_dict[name])
    root=stack[0]
    res=Tree_BFS(root,l)
    return res

# 用BFS的方法遍历二叉树
def Tree_BFS(root,l):
    queue=deque()
    queue.append(root)
    res=[root.value]
    while len(res)<l:
        a=queue.popleft()
        if a.left!=None:
            res.append(a.left.value)
            queue.append(a.left)
        if a.right!=None:
            res.append(a.right.value)
            queue.append(a.right)
    return res

for _ in range(int(input())):
    raw=deque(input())
    print("".join(reversed(Parse_tree(raw))))
```

基本信息

#: 44282867
题目: 25140
提交人: 23n2300011119 (武)
内存: 3716kB
时间: 30ms
语言: Python3
提交时间: 2024-03-18 10:13:54

English 帮助 关于

24750: 根据二叉树中后序序列建树

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

思路:

注：这里是"猜二叉树"的代码，题目除了输入是完全一样的。

代码

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

#初始化global变量
node_dict,pre_order,idx,current_node=dict(),[],0,None

# 函数的功能是建立起以name为根的子树，参数是name和中序表达式
def TreeBuilding(name,in_order:list):
    # idx全局变量寻找左子树根
    # current_node指向现在操作的对象
```

```
    global idx,current_node,node_dict,pre_order
    #设置递归出口
    if len(in_order)==1:
        node_dict[name]=TreeNode(name)
        if current_node.left==None:
            current_node.left=node_dict[name]
            return
        current_node.right=node_dict[name]
        return

    # 建立树根并存在字典中，便于索引
    node_dict[name]=TreeNode(name)

    # 如果name节点是一个子节点，那current_node!=None
    # 建立起name和current_node的连接。
    if current_node!=None:
        if current_node.left==None:
            current_node.left=node_dict[name]
            pass
        elif current_node.right==None:
            current_node.right=node_dict[name]

    # 标明现在状态
    current_node=node_dict[name]
    pivot=in_order.index(name)

    # 建立右子树
    ltree_in_order=in_order[:pivot]
    if ltree_in_order:
        idx+=1
        TreeBuilding(pre_order[idx],ltree_in_order)

    # 建立右子树
    current_node=node_dict[name]
    rtree_in_order=in_order[pivot+1:]
    if rtree_in_order:
        idx+=1
        TreeBuilding(pre_order[idx],rtree_in_order)

def post_search(root):
    if root==None:
        return ""
    output=[]
    output.extend(post_search(root.left))
    output.extend(post_search(root.right))
    output.append(root.value)
    return "".join(output)

while True:
    try:
        node_dict = dict()
        pre_order = list(input())
        in_order = list(input())
        # 最初的父节点指向None，即根节点的父节点指向None
        current_node = None
        idx = 0
```

```
        if len(pre_order) == 1:
            print(pre_order[0])
        else:
            TreeBuilding(pre_order[idx], in_order)
            print(post_search(node_dict[pre_order[0]]))
    except EOFError:
        break
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44205844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

#初始化global变量
node_dict,pre_order,idx,current_node=dict(),[],0,None

# 函数的功能是建立起以name为根的子树，参数是name和中序表达式
def TreeBuilding(name,in_order:list):
    # idx全局变量寻找左子树根
    # current_node指向现在操作的对象
    global idx,current_node,node_dict,pre_order
    #设置递归出口
    if len(in_order)==1:
        node_dict[name]=TreeNode(name)
        if current_node.left==None:
            current_node.left=node_dict[name]
            return
        current_node.right=node_dict[name]
        return

    # 建立树根并存在字典中，便于索引
```

基本信息

#: 44205844
题目: 22158
提交人: 23n2300011119（武）
内存: 3660kB
时间: 22ms
语言: Python3
提交时间: 2024-03-13 22:22:20

22158: 根据二叉树前中序序列建树

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

思路:

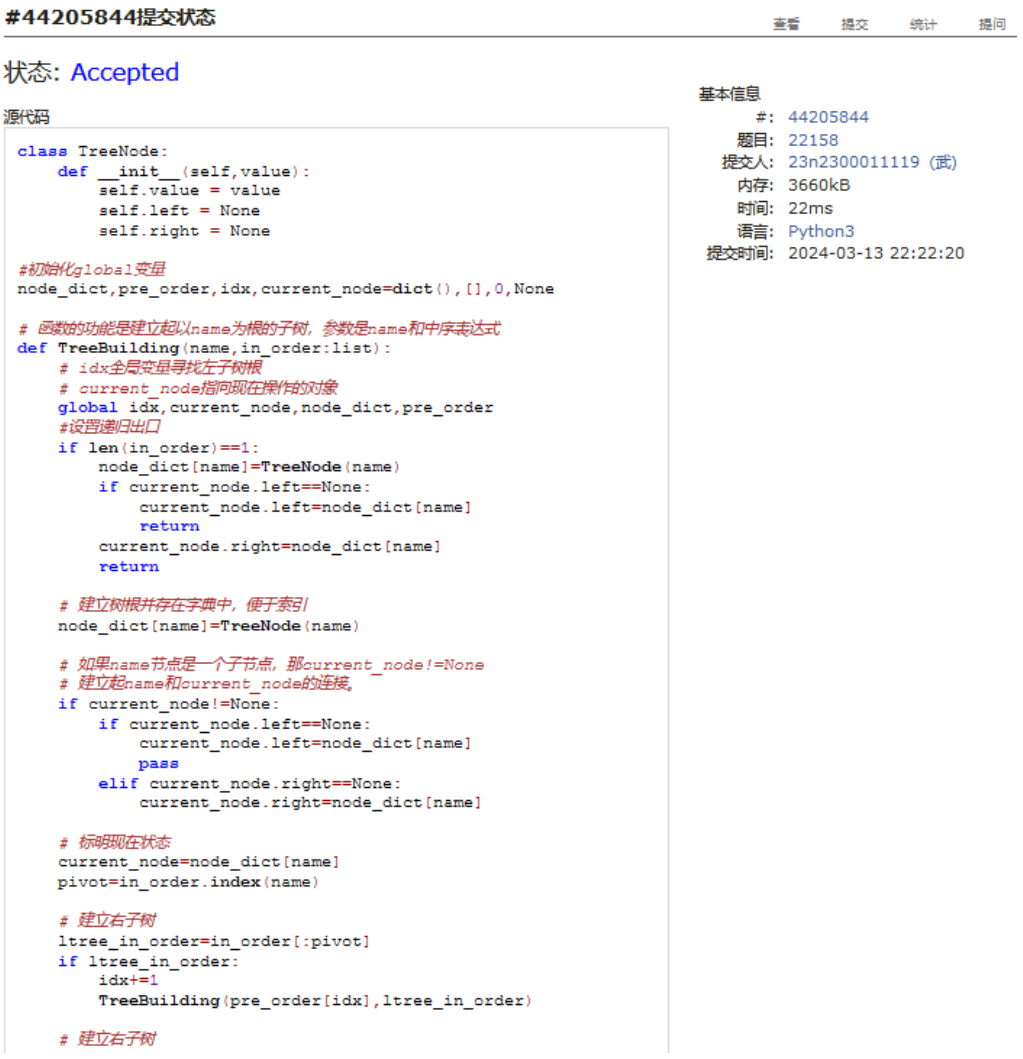
代码

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

#初始化global变量
node_dict,pre_order,idx,current_node=dict(),[],0,None

# 函数的功能是建立起以name为根的子树，参数是name和中序表达式
def TreeBuilding(name,in_order:list):
    # idx全局变量寻找左子树根
    # current_node指向现在操作的对象
    global idx,current_node,node_dict,pre_order
    #设置递归出口
    if len(in_order)==1:
        node_dict[name]=TreeNode(name)
        if current_node.left==None:
```

```
            current_node.left=node_dict[name]
            return
        current_node.right=node_dict[name]
        return

    # 建立树根并存在字典中，便于索引
    node_dict[name]=TreeNode(name)

    # 如果name节点是一个子节点，那current_node!=None
    # 建立起name和current_node的连接。
    if current_node!=None:
        if current_node.left==None:
            current_node.left=node_dict[name]
            pass
        elif current_node.right==None:
            current_node.right=node_dict[name]

    # 标明现在状态
    current_node=node_dict[name]
    pivot=in_order.index(name)

    # 建立右子树
    ltree_in_order=in_order[:pivot]
    if ltree_in_order:
        idx+=1
        TreeBuilding(pre_order[idx],ltree_in_order)

    # 建立右子树
    current_node=node_dict[name]
    rtree_in_order=in_order[pivot+1:]
    if rtree_in_order:
        idx+=1
        TreeBuilding(pre_order[idx],rtree_in_order)

def post_search(root):
    if root==None:
        return ""
    output=[]
    output.extend(post_search(root.left))
    output.extend(post_search(root.right))
    output.append(root.value)
    return "".join(output)

while True:
    try:
        node_dict = dict()
        pre_order = list(input())
        in_order = list(input())
        # 最初的父节点指向None，即根节点的父节点指向None
        current_node = None
        idx = 0
        if len(pre_order) == 1:
            print(pre_order[0])
        else:
            TreeBuilding(pre_order[idx], in_order)
            print(post_search(node_dict[pre_order[0]]))
```

```
    except EOFError:
        break
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44205844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

#初始化global变量
node_dict,pre_order,idx,current_node=dict(),[],0,None

# 函数的功能是建立起以name为根的子树，参数是name和中序表达式
def TreeBuilding(name,in_order:list):
    # idx全局变量寻找左子树根
    # current_node指向现在操作的对象
    global idx,current_node,node_dict,pre_order
    #设置递归出口
    if len(in_order)==1:
        node_dict[name]=TreeNode(name)
        if current_node.left==None:
            current_node.left=node_dict[name]
            return
        current_node.right=node_dict[name]
        return

    # 建立树根并存在字典中，便于索引
    node_dict[name]=TreeNode(name)

    # 如果name节点是一个子节点，那current_node!=None
    # 建立起name和current_node的连接。
    if current_node!=None:
        if current_node.left==None:
            current_node.left=node_dict[name]
            pass
        elif current_node.right==None:
            current_node.right=node_dict[name]

    # 标明现在状态
    current_node=node_dict[name]
    pivot=in_order.index(name)

    # 建立右子树
    ltree_in_order=in_order[:pivot]
    if ltree_in_order:
        idx+=1
        TreeBuilding(pre_order[idx],ltree_in_order)

    # 建立右子树
```

基本信息

#: 44205844
题目: 22158
提交人: 23n2300011119 (武)
内存: 3660kB
时间: 22ms
语言: Python3
提交时间: 2024-03-13 22:22:20

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。

做完每日选做会感受到树的题目大同小异，考察点在于**类，引用与递归**，并且代码可复用性强。

最近几天的每日选做（现在是3.18）都跟递归有关，前一段时间练习完树的题目后发现写递归变得相当轻松，上学期的重大障碍被攻克了。