

# Assignment #5: "树"算：概念、表示、解析、遍历

---

Updated 2124 GMT+8 March 17, 2024

2024 spring, Compiled by 刘思瑞 2100017810

## 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：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：

写一个类然后二叉树遍历

代码

```
1 class TreeNode:
2     def __init__(self):
3         self.left = None
4         self.right = None
5     def tree_height(node):
6         if node is None:
```

```

7         return -1
8     return max(tree_height(node.left), tree_height(node.right)) + 1
9 def count_leaves(node):
10     if node is None:
11         return 0
12     if node.left is None and node.right is None:
13         return 1
14     return count_leaves(node.left) + count_leaves(node.right)
15 n = int(input())
16 nodes = [TreeNode() for _ in range(n)]
17 has_parent = [False] * n
18 for i in range(n):
19     left_index, right_index = map(int, input().split())
20     if left_index != -1:
21         nodes[i].left = nodes[left_index]
22         has_parent[left_index] = True
23     if right_index != -1:
24         #print(right_index)
25         nodes[i].right = nodes[right_index]
26         has_parent[right_index] = True
27 root_index = has_parent.index(False)
28 root = nodes[root_index]
29 height = tree_height(root)
30 leaves = count_leaves(root)
31 print(f"{height} {leaves}")

```

代码运行截图

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)
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:

```

## 24729: 括号嵌套树

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

思路:

用括号判断进出栈

代码

```
1  def parse_tree(s):
2      stack = []
3      node = None
4      for char in s:
5          if char.isalpha():
6              node = TreeNode(char)
7              if stack:
8                  stack[-1].children.append(node)
9          elif char == '(':
10             if node:
11                 stack.append(node)
12                 node = None
13          elif char == ')':
14             if stack:
15                 node = stack.pop()
16     return node
17
18
19 def preorder(node):
20     output = [node.value]
21     for child in node.children:
22         output.extend(preorder(child))
23     return ''.join(output)
24
25 def postorder(node):
26     output = []
27     for child in node.children:
28         output.extend(postorder(child))
29     output.append(node.value)
30     return ''.join(output)
31
32 def main():
33     s = input().strip()
34     s = ''.join(s.split())
35     root = parse_tree(s)
36     if root:
37         print(preorder(root))
38         print(postorder(root))
39     else:
40         print("input tree string error!")
41
42 if __name__ == "__main__":
43     main()
```

代码运行截图

#000015122提交记录

状态: Accepted

源代码

```
def parse_tree(s):
    stack = []
    node = None
    for char in s:
        if char.isalpha():
```

## 02775: 文件结构“图”

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

思路:

代码

```
1  from sys import exit
2
3  class dir:
4      def __init__(self, dname):
5          self.name = dname
6          self.dirs = []
7          self.files = []
8
9      def getGraph(self):
10         g = [self.name]
11         for d in self.dirs:
12             subg = d.getGraph()
13             g.extend(["| " + s for s in subg])
14         for f in sorted(self.files):
15             g.append(f)
16         return g
17
18  n = 0
19  while True:
20      n += 1
21      stack = [dir("ROOT")]
22      while (s := input()) != "":
23          if s == "#": exit(0)
24          if s[0] == 'f':
```

```

25         stack[-1].files.append(s)
26     elif s[0] == 'd':
27         stack.append(dir(s))
28         stack[-2].dirs.append(stack[-1])
29     else:
30         stack.pop()
31     print(f"DATA SET {n}:")
32     print(*stack[0].getGraph(), sep='\n')
33     print()

```

代码运行截图

状态: Accepted

源代码

```

from sys import exit

class dir:
    def __init__(self, dname):
        self.name = dname
        self.dirs = []
        self.files = []

    def getGraph(self):
        g = [self.name]
        for d in self.dirs:

```

## 25140: 根据后序表达式建立队列表达式

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

思路:

遍历树的结构

代码

```

1 class TreeNode:
2     def __init__(self, value):
3         self.value = value
4         self.left = None
5         self.right = None
6
7 def build_tree(postfix):
8     stack = []
9     for char in postfix:
10         node = TreeNode(char)
11         if char.isupper():
12             node.right = stack.pop()

```

```

13         node.left = stack.pop()
14         stack.append(node)
15     return stack[0]
16
17 def level_order_traversal(root):
18     queue = [root]
19     traversal = []
20     while queue:
21         node = queue.pop(0)
22         traversal.append(node.value)
23         if node.left:
24             queue.append(node.left)
25         if node.right:
26             queue.append(node.right)
27     return traversal
28
29 n = int(input().strip())
30 for _ in range(n):
31     postfix = input().strip()
32     root = build_tree(postfix)
33     queue_expression = level_order_traversal(root)[::-1]
34     print(''.join(queue_expression))

```

代码运行截图

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):
    -
    -

```

## 24750: 根据二叉树中后序序列建树

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

思路:

代码

```
1 def build_tree(inorder, postorder):
2     if not inorder or not postorder:
3         return []
4
5     root_val = postorder[-1]
6     root_index = inorder.index(root_val)
7
8     left_inorder = inorder[:root_index]
9     right_inorder = inorder[root_index + 1:]
10
11     left_postorder = postorder[:len(left_inorder)]
12     right_postorder = postorder[len(left_inorder):-1]
13
14     root = [root_val]
15     root.extend(build_tree(left_inorder, left_postorder))
16     root.extend(build_tree(right_inorder, right_postorder))
17
18     return root
19
20
21 def main():
22     inorder = input().strip()
23     postorder = input().strip()
24     preorder = build_tree(inorder, postorder)
25     print(' '.join(preorder))
26
27
28 if __name__ == "__main__":
29     main()
```

代码运行截图

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))
```

## 22158: 根据二叉树前中序序列建树

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

思路:

代码

```
1 class TreeNode:
2     def __init__(self, value):
3         self.value = value
4         self.left = None
5         self.right = None
6
7 def build_tree(preorder, inorder):
8     if not preorder or not inorder:
9         return None
10    root_value = preorder[0]
11    root = TreeNode(root_value)
12    root_index_inorder = inorder.index(root_value)
```



```

13     root.left = build_tree(preorder[1:1+root_index_inorder],
14                             inorder[:root_index_inorder])
15     root.right = build_tree(preorder[1+root_index_inorder:],
16                             inorder[root_index_inorder+1:])
17     return root
18
19 def postorder_traversal(root):
20     if root is None:
21         return ''
22     return postorder_traversal(root.left) + postorder_traversal(root.right) +
23         root.value
24
25 while True:
26     try:
27         preorder = input().strip()
28         inorder = input().strip()
29         root = build_tree(preorder, inorder)
30         print(postorder_traversal(root))
31     except EOFError:
32         break

```

代码运行截图

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

```

## 2. 学习总结和收获

这周太忙了没来得及做

