Assignment #8: 树为主

Updated 1704 GMT+8 Apr 8, 2025

2025 spring, Complied by **袁奕 2400010766 数院**

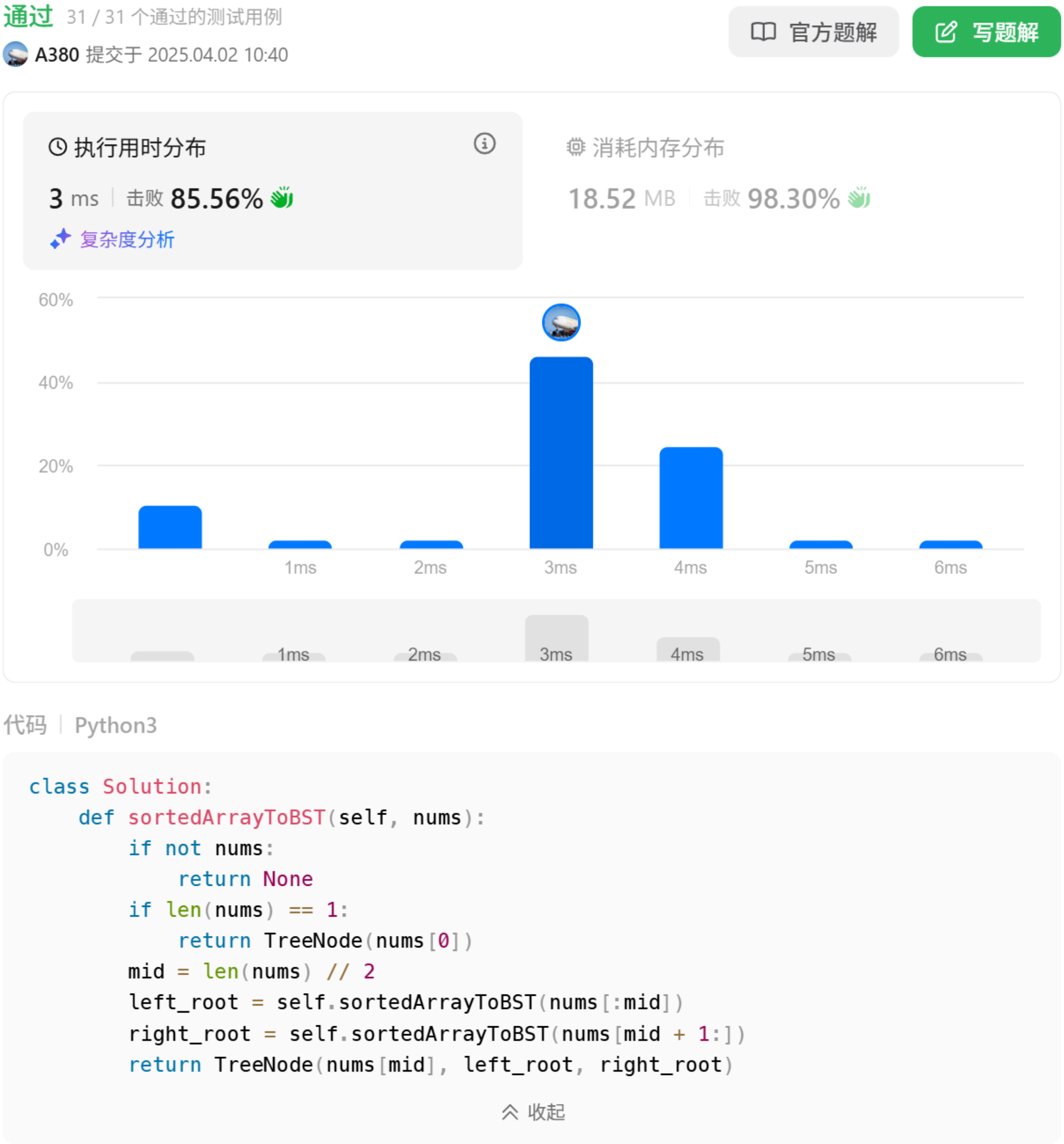
> **说明：**
>
> 1. **解题与记录：**
>
>   对于每一个题目，请提供其解题思路（可选），并附上使用Python或C++编写的源代码（确保已在OpenJudge， Codeforces，LeetCode等平台上获得Accepted）。请将这些信息连同显示“Accepted”的截图一起填写到下方的作业模板中。（推荐使用Typora https://typoraio.cn 进行编辑，当然你也可以选择Word。）无论题目是否已通过，请标明每个题目大致花费的时间。
>
> 2. **提交安排：**提交时，请首先上传PDF格式的文件，并将.md或.doc格式的文件作为附件上传至右侧的“作业评论”区。确保你的Canvas账户有一个清晰可见的头像，提交的文件为PDF格式，并且“作业评论”区包含上传的.md或.doc附件。
>
> 3. **延迟提交：**如果你预计无法在截止日期前提交作业，请提前告知具体原因。这有助于我们了解情况并可能为你提供适当的延期或其他帮助。
>
> 请按照上述指导认真准备和提交作业，以保证顺利完成课程要求。

1. 题目

LC108.将有序数组转换为二叉树

dfs, https://leetcode.cn/problems/convert-sorted-array-to-binary-search-tree/

```python
class Solution:
    def sortedArrayToBST(self, nums):
        if not nums:
            return None
        if len(nums) == 1:
            return TreeNode(nums[0])
        mid = len(nums) // 2
        left_root = self.sortedArrayToBST(nums[:mid])
        right_root = self.sortedArrayToBST(nums[mid + 1:])
        return TreeNode(nums[mid], left_root, right_root)
```



通过 31 / 31 个通过的测试用例

A380 提交于 2025.04.02 10:40

执行用时分布

3 ms | 击败 85.56%

消耗内存分布

18.52 MB | 击败 98.30%

代码 | Python3

```python
class Solution:
    def sortedArrayToBST(self, nums):
        if not nums:
            return None
        if len(nums) == 1:
            return TreeNode(nums[0])
        mid = len(nums) // 2
        left_root = self.sortedArrayToBST(nums[:mid])
        right_root = self.sortedArrayToBST(nums[mid + 1:])
        return TreeNode(nums[mid], left_root, right_root)
```

M27928:遍历树

adjacency list, dfs, http://cs101.openjudge.cn/practice/27928/

思路：中间加了`val_node`来存储值对应的`TreeNode`节点, 感觉比较麻烦, 是否可以简略?

以及这种写法会有问题

```python
class TreeNode(object):
    def init(self, val=0, children = []):
        self.val = val
        self.children = children
```

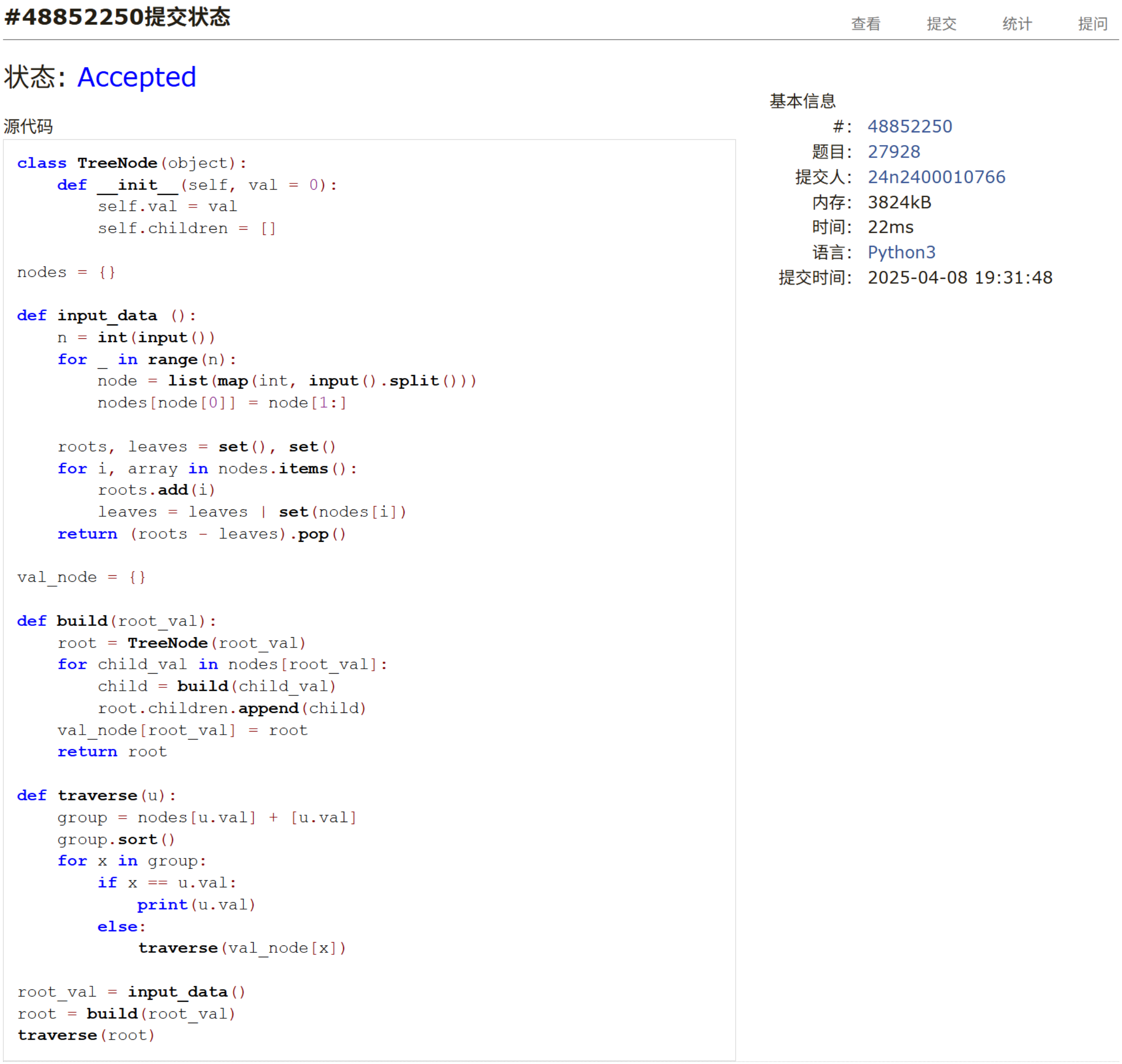
ChatGPT said:

这段代码存在一个潜在的**问题**，即使用了可变对象（如列表）作为默认参数值。

具体来说，`children = []` 这种写法可能会导致不可预期的行为。因为 Python 中的默认参数值是在函数定义时评估的，因此对于可变对象（如列表、字典等），它只会创建一次，并且在函数调用过程中如果修改了这个对象，它会影响到后续的函数调用。

具体到你的代码，假设你创建了多个 `TreeNode` 实例并没有传递 `children` 参数，所有这些实例的 `children` 会指向同一个列表对象，而不是每次都创建一个新的列表。这样，当你修改一个节点的 `children` 时，其他所有节点的 `children` 也会被修改。

```python
class TreeNode(object):
    def __init__(self, val = 0):
        self.val = val
        self.children = []

nodes = {}

def input_data ():
    n = int(input())
    for _ in range(n):
        node = list(map(int, input().split()))
        nodes[node[0]] = node[1:]

    roots, leaves = set(), set()
    for i, array in nodes.items():
        roots.add(i)
        leaves = leaves | set(nodes[i])
    return (roots - leaves).pop()

val_node = {}

def build(root_val):
    root = TreeNode(root_val)
    for child_val in nodes[root_val]:
        child = build(child_val)
        root.children.append(child)
    val_node[root_val] = root
    return root

def traverse(u):
    group = nodes[u.val] + [u.val]
    group.sort()
    for x in group:
        if x == u.val:
            print(u.val)
        else:
            traverse(val_node[x])

root_val = input_data()
root = build(root_val)
traverse(root)
```



#48852250提交状态

状态: Accepted

源代码

```
class TreeNode(object):
    def __init__(self, val = 0):
        self.val = val
        self.children = []

nodes = {}

def input_data ():
    n = int(input())
    for _ in range(n):
        node = list(map(int, input().split()))
        nodes[node[0]] = node[1:]

    roots, leaves = set(), set()
    for i, array in nodes.items():
        roots.add(i)
        leaves = leaves | set(nodes[i])
    return (roots - leaves).pop()

val_node = {}

def build(root_val):
    root = TreeNode(root_val)
    for child_val in nodes[root_val]:
        child = build(child_val)
        root.children.append(child)
    val_node[root_val] = root
    return root

def traverse(u):
    group = nodes[u.val] + [u.val]
    group.sort()
    for x in group:
        if x == u.val:
            print(u.val)
        else:
            traverse(val_node[x])

root_val = input_data()
root = build(root_val)
traverse(root)
```

基本信息

#:	48852250
题目:	27928
提交人:	24n2400010766
内存:	3824kB
时间:	22ms
语言:	Python3
提交时间:	2025-04-08 19:31:48

LC129.求根节点到叶节点数字之和

dfs, https://leetcode.cn/problems/sum-root-to-leaf-numbers/

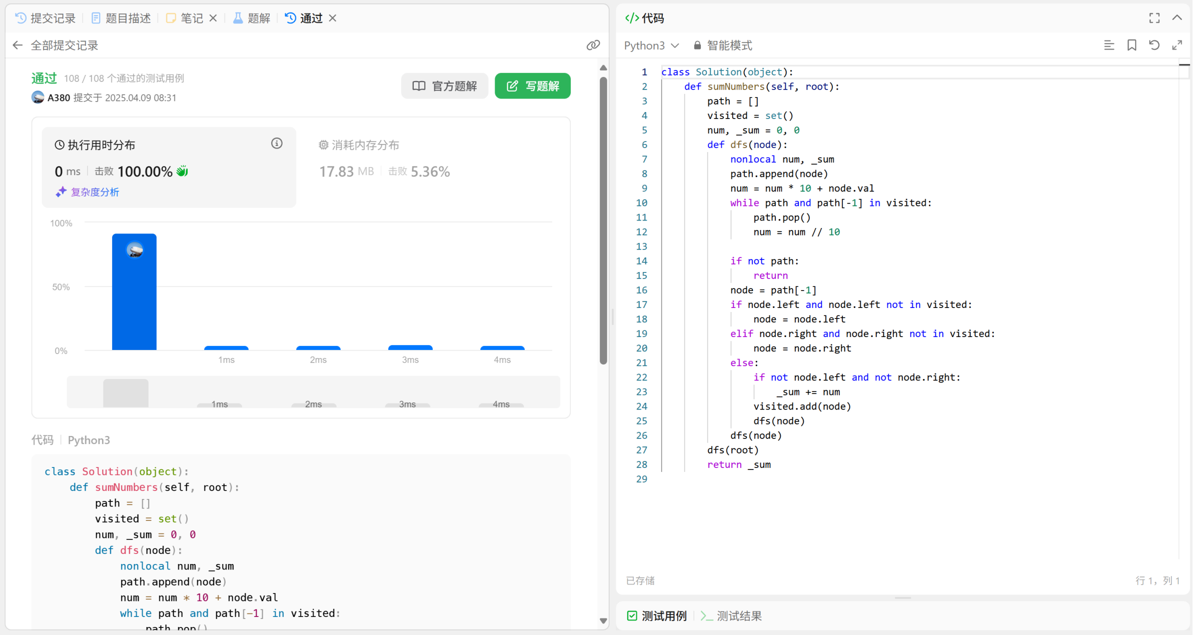
思路：学习了 `nonlocal` 的用法.

方法仿照 230. 二叉搜索树中第 K 小的元素, 即 `path` 记录路径, 遇到叶子节点或者已经搜索过的分支后用 `visited` 标记并沿 `path` 回溯

看题解了解到, 类似递归的方式可能更简单

```
class Solution(object):
    def sumNumbers(self, root):
        path = []
        visited = set()
        num, _sum = 0, 0
        def dfs(node):
            nonlocal num, _sum
            path.append(node)
            num = num * 10 + node.val
            while path and path[-1] in visited:
                path.pop()
                num = num // 10

            if not path:
```

```python
                return
            node = path[-1]
            if node.left and node.left not in visited:
                node = node.left
            elif node.right and node.right not in visited:
                node = node.right
            else:
                if not node.left and not node.right:
                    _sum += num
                visited.add(node)
                dfs(node)
            dfs(node)
        dfs(root)
        return _sum
```



通过 108 / 108 个通过的测试用例

A380 提交于 2025.04.09 08:31

执行用时分布 0 ms | 击败 100.00%

消耗内存分布 17.83 MB | 击败 5.36%

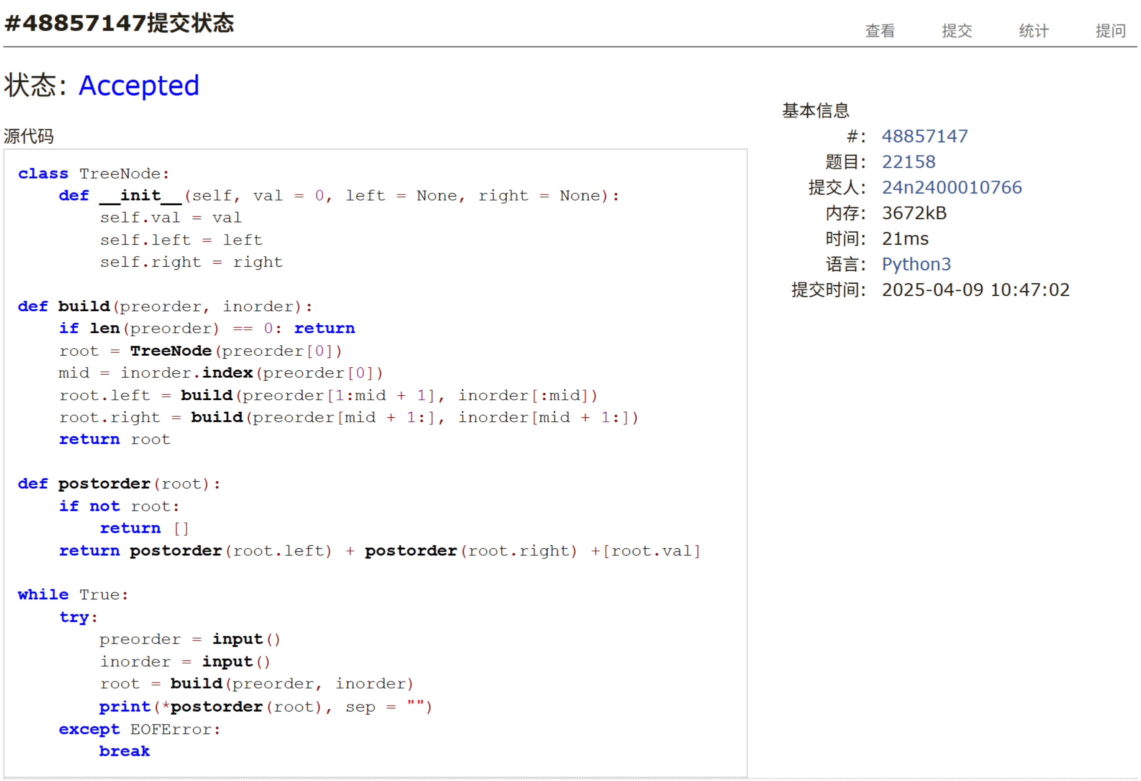
代码 | Python3

```python
class Solution(object):
    def sumNumbers(self, root):
        path = []
        visited = set()
        num, _sum = 0, 0
        def dfs(node):
            nonlocal num, _sum
            path.append(node)
            num = num * 10 + node.val
            while path and path[-1] in visited:
                path.pop()
                num = num // 10

            if not path:
                return
            node = path[-1]
            if node.left and node.left not in visited:
                node = node.left
            elif node.right and node.right not in visited:
                node = node.right
            else:
                if not node.left and not node.right:
                    _sum += num
                visited.add(node)
                dfs(node)
            dfs(node)
        dfs(root)
        return _sum
```

M22158:根据二叉树前中序序列建树

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

```python
class TreeNode:
    def __init__(self, val = 0, left = None, right = None):
        self.val = val
        self.left = left
        self.right = right

def build(preorder, inorder):
    if len(preorder) == 0: return
    root = TreeNode(preorder[0])
    mid = inorder.index(preorder[0])
    root.left = build(preorder[1:mid + 1], inorder[:mid])
    root.right = build(preorder[mid + 1:], inorder[mid + 1:])
    return root

def postorder(root):
    if not root:
        return []
    return postorder(root.left) + postorder(root.right) +[root.val]

while True:
    try:
        preorder = input()
        inorder = input()
        root = build(preorder, inorder)
        print(*postorder(root), sep = "")
    except EOFError:
        break
```



#48857147提交状态

状态: Accepted

源代码

```
class TreeNode:
    def __init__(self, val = 0, left = None, right = None):
        self.val = val
        self.left = left
        self.right = right

def build(preorder, inorder):
    if len(preorder) == 0: return
    root = TreeNode(preorder[0])
    mid = inorder.index(preorder[0])
    root.left = build(preorder[1:mid + 1], inorder[:mid])
    root.right = build(preorder[mid + 1:], inorder[mid + 1:])
    return root

def postorder(root):
    if not root:
        return []
    return postorder(root.left) + postorder(root.right) +[root.val]

while True:
    try:
        preorder = input()
        inorder = input()
        root = build(preorder, inorder)
        print(*postorder(root), sep = "")
    except EOFError:
        break
```

基本信息

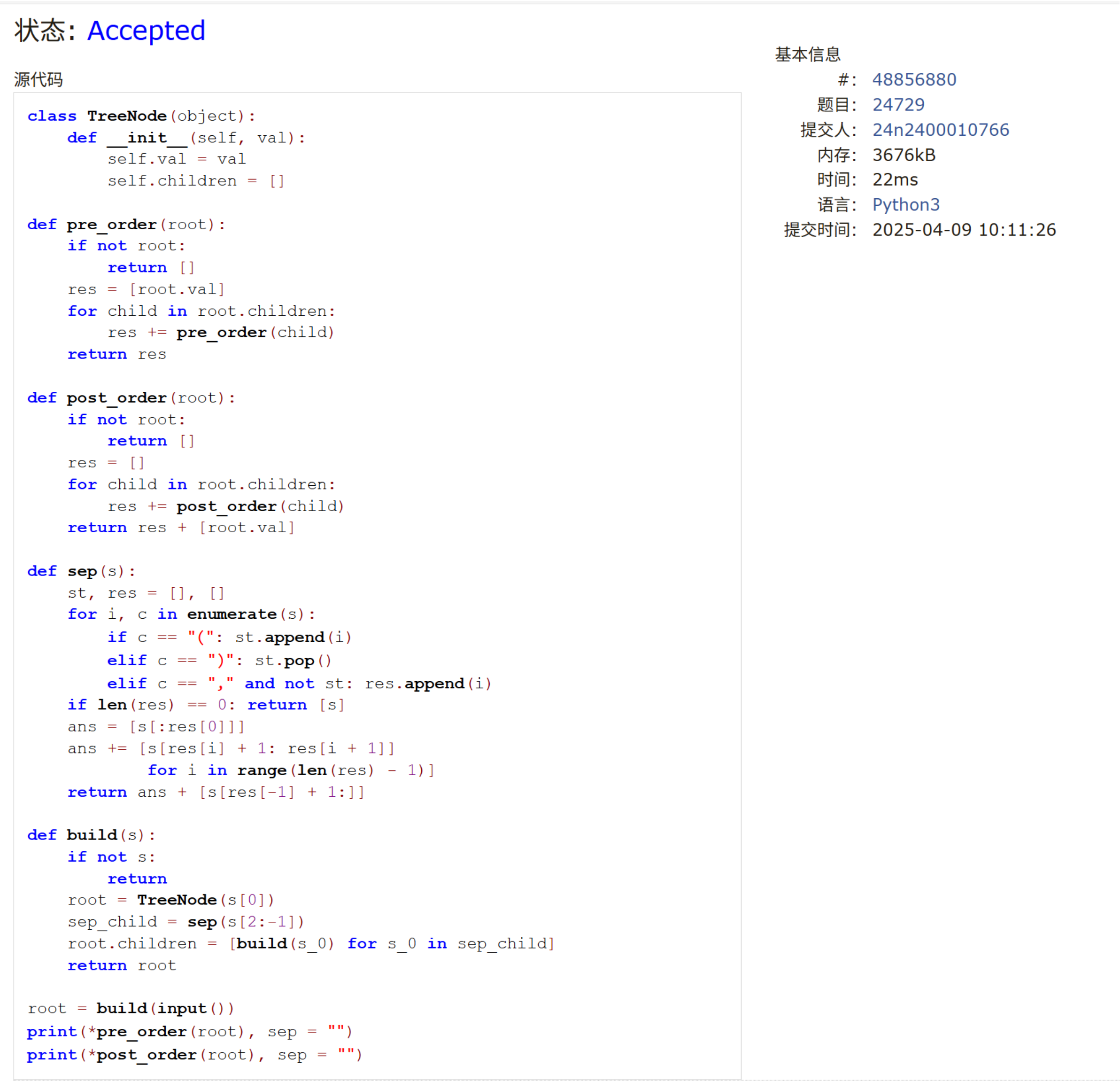
#: 48857147
题目: 22158
提交人: 24n2400010766
内存: 3672kB
时间: 21ms
语言: Python3
提交时间: 2025-04-09 10:47:02

T24729:括号嵌套树

dfs, stack, http://cs101.openjudge.cn/practice/24729/

思路：其中`sep`用来划分子节点, 但是感觉分类讨论有些冗长

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

def pre_order(root):
    if not root:
        return []
    res = [root.val]
    for child in root.children:
        res += pre_order(child)
    return res

def post_order(root):
    if not root:
        return []
    res = []
    for child in root.children:
        res += post_order(child)
    return res + [root.val]

def sep(s):
    st, res = [], []
    for i, c in enumerate(s):
        if c == "(": st.append(i)
        elif c == ")": st.pop()
        elif c == "," and not st: res.append(i)
    if len(res) == 0: return [s]
    ans = [s[:res[0]]]
    ans += [s[res[i] + 1: res[i + 1]]
            for i in range(len(res) - 1)]
    return ans + [s[res[-1] + 1:]]

def build(s):
    if not s:
        return
    root = TreeNode(s[0])
    sep_child = sep(s[2:-1])
    root.children = [build(s_0) for s_0 in sep_child]
    return root

root = build(input())
print(*pre_order(root), sep = "")
print(*post_order(root), sep = "")
```


状态: Accepted

源代码

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

def pre_order(root):
    if not root:
        return []
    res = [root.val]
    for child in root.children:
        res += pre_order(child)
    return res

def post_order(root):
    if not root:
        return []
    res = []
    for child in root.children:
        res += post_order(child)
    return res + [root.val]

def sep(s):
    st, res = [], []
    for i, c in enumerate(s):
        if c == "(": st.append(i)
        elif c == ")": st.pop()
        elif c == "," and not st: res.append(i)
    if len(res) == 0: return [s]
    ans = [s[:res[0]]]
    ans += [s[res[i] + 1: res[i + 1]]
           for i in range(len(res) - 1)]
    return ans + [s[res[-1] + 1:]]

def build(s):
    if not s:
        return
    root = TreeNode(s[0])
    sep_child = sep(s[2:-1])
    root.children = [build(s_0) for s_0 in sep_child]
    return root

root = build(input())
print(*pre_order(root), sep = "")
print(*post_order(root), sep = "")
```

基本信息

#:	48856880
题目:	24729
提交人:	24n2400010766
内存:	3676kB
时间:	22ms
语言:	Python3
提交时间:	2025-04-09 10:11:26

LC3510.移除最小数对使数组有序II

doubly-linked list + heap, https://leetcode.cn/problems/minimum-pair-removal-to-sort-array-ii/

思路：耗时 6h, 从下午 16:00 ~ 晚上 21:00, 中途吃了顿饭 (~~独立做出, 快夸我(doge)~~)

首先自然的想到用 `doubly-linked list` 和 `heap` 懒删除模拟维护

问题0. 懒删除带来的自然问题, 如何判断 `hp` 中的元素是否失效

解决. `heap` 中维护 `(sum, id)`, 比较 `sum = node.val + node.next.val` 即可. 其中 `node` 为 `id` 位置处合并所得

问题1. 每次操作 (merge) 后都需要判断是否单调不减

解决. 用 `cnt` 存储相邻对中的逆序对, 这样每次 merge 后可以迅速的 update (只用进行 $O(1)$ 的判断)

并且当 `cnt == 0` 时自动是单调不减的

问题2. `node` 和 `node.next` merge 后如何快速找到 `node.pre` 的下标呢?

于是引入了 `id` 作为下标的双头链表

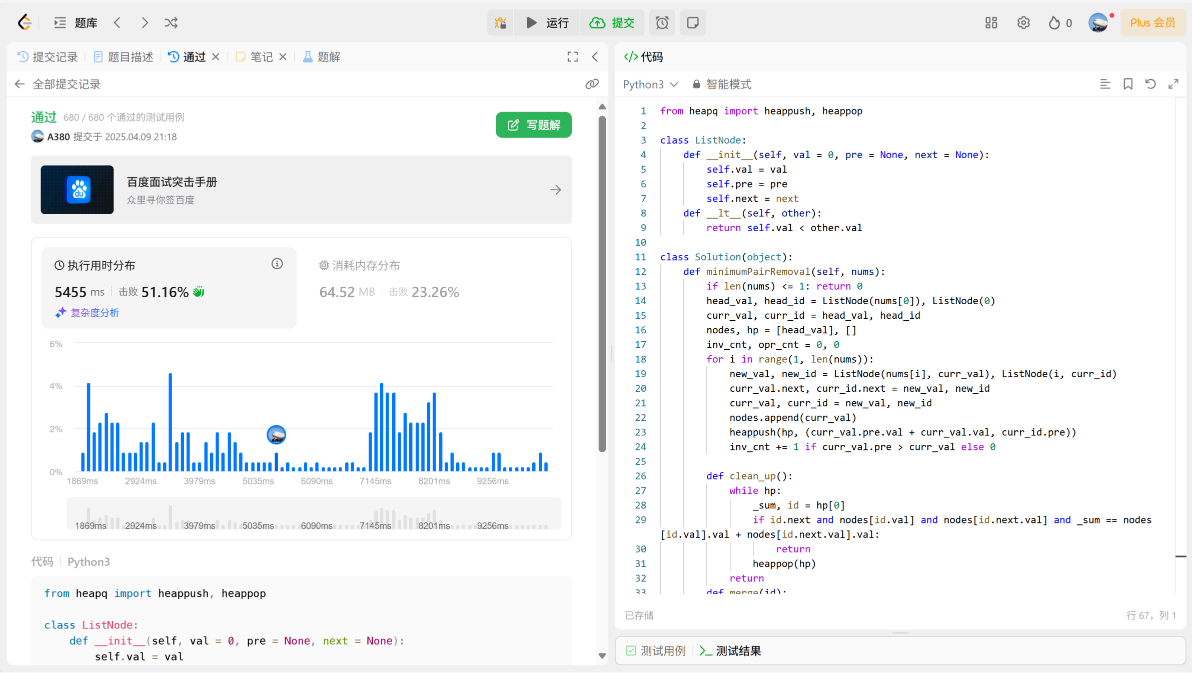
问题3. 写完调试发现, `sum` 相同时, 并不一定按照 `id` 的大小排序

解决. 给 `ListNode` 加入 `__lt__` 的OOP

问题4. 边界情况的验证繁琐复杂, 调试了非常长时间

代码:

```python
from heapq import heappush, heappop

class ListNode:
    def __init__(self, val = 0, pre = None, next = None):
        self.val = val
        self.pre = pre
        self.next = next
    def __lt__(self, other):
        return self.val < other.val

class Solution(object):
    def minimumPairRemoval(self, nums):
        if len(nums) <= 1: return 0
        head_val, head_id = ListNode(nums[0]), ListNode(0)
        curr_val, curr_id = head_val, head_id
        nodes, hp = [head_val], []
        inv_cnt, opr_cnt = 0, 0

        # init
        for i in range(1, len(nums)):
            new_val, new_id = ListNode(nums[i], curr_val), ListNode(i, curr_id)
            curr_val.next, curr_id.next = new_val, new_id
            curr_val, curr_id = new_val, new_id
            nodes.append(curr_val)
            # hp 中维护 (nodes[i] + nodes[i.next], i)
            heappush(hp, (curr_val.pre.val + curr_val.val, curr_id.pre))
            inv_cnt += 1 if curr_val.pre > curr_val else 0

        # 懒删除
        def clean_up():
            while hp:
                _sum, id = hp[0]
                if id.next and nodes[id.val] and nodes[id.next.val] and _sum == nodes[id.val].val + nodes[id.next.val].val:
                    return
                heappop(hp)
            return

        def merge(id):
            nonlocal inv_cnt
            if nodes[id.val] > nodes[id.next.val]: inv_cnt -= 1
            if id.pre and nodes[id.pre.val] > nodes[id.val]:
                inv_cnt -= 1
            if (id.next and id.next.next and
                nodes[id.next.val] > nodes[id.next.next.val]):
                inv_cnt -= 1

            # 将 (node[i], i) 和 (node[i.next], i.next) 合并为 (node[i] + node[i.next], i)
            nodes[id.val].val = nodes[id.val].val + nodes[id.next.val].val
            nodes[id.next.val] = None
            nodes[id.val].next = nodes[id.val].next.next

            id.next = id.next.next

            if id.next:
                nodes[id.val].next.pre = nodes[id.val]
                id.next.pre = id

            if id.next and nodes[id.val] > nodes[id.next.val]: inv_cnt += 1
            if id.pre and nodes[id.pre.val] > nodes[id.val]:
                inv_cnt += 1

            if id.next:
                heappush(hp, (nodes[id.val].val + nodes[id.next.val].val, id))
            if id.pre:
                heappush(hp, (nodes[id.pre.val].val + nodes[id.val].val, id.pre))

        while inv_cnt != 0:
            opr_cnt += 1
            clean_up()
            _, id = heappop(hp)
            merge(id)
        return opr_cnt
```



通过 680 / 680 个通过的测试用例

A380 提交于 2025.04.09 21:18

5455 ms | 击败 51.16%

64.52 MB | 击败 23.26%

代码 | Python3

```
from heapq import heappush, heappop

class ListNode:
    def __init__(self, val = 0, pre = None, next = None):
        self.val = val
        self.pre = pre
        self.next = next
    def __lt__(self, other):
        return self.val < other.val

class Solution(object):
    def minimumPairRemoval(self, nums):
        if len(nums) <= 1: return 0
        head_val, head_id = ListNode(nums[0]), ListNode(0)
        curr_val, curr_id = head_val, head_id
        nodes, hp = [head_val], []
        inv_cnt, opr_cnt = 0, 0
        for i in range(1, len(nums)):
            new_val, new_id = ListNode(nums[i], curr_val), ListNode(i, curr_id)
            curr_val.next, curr_id.next = new_val, new_id
            curr_val, curr_id = new_val, new_id
            nodes.append(curr_val)
            heappush(hp, (curr_val.pre.val + curr_val.val, curr_id.pre))
            inv_cnt += 1 if curr_val.pre > curr_val else 0

        def clean_up():
            while hp:
                _sum, id = hp[0]
                if id.next and nodes[id.val] and nodes[id.next.val] and _sum == nodes[id.val].val + nodes[id.next.val].val:
                    return
                heappop(hp)
            return
```

2. 学习总结和收获

最近过的浑浑噩噩的, 先想办法活着挺过期中季……

画大饼：之后想着重练习类似3510. 移除最小数对使数组有序 II 这样要求多种数据结构组合使用的题目. 例如 heap + 懒删除 + linked list.

以及感觉自己写代码喜欢分类讨论, 导致代码及其冗长, 中间出现 typo 的概率也更高. 希望以后通过学习题解 / gpt 等方式精简代码.