1. 题目

LC222.完全二叉树的节点个数

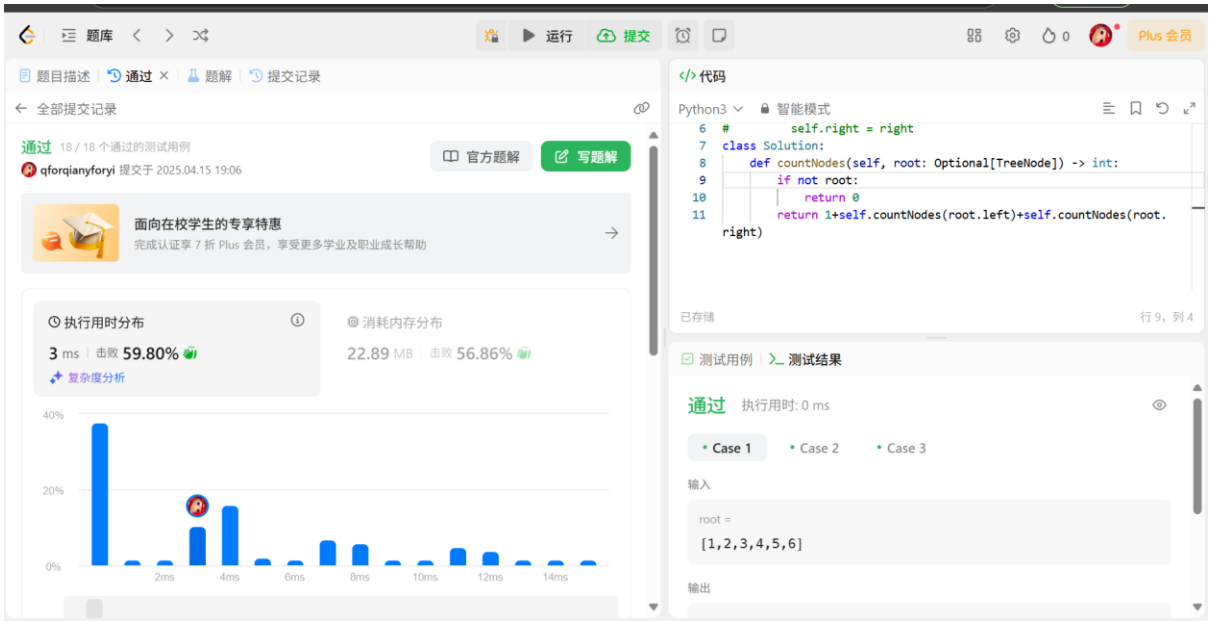
dfs, https://leetcode.cn/problems/count-complete-tree-nodes/

思路：递归，每次检查左右节点，如果有节点就加一，否则返回 0

代码：

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def countNodes(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0
        return 1+self.countNodes(root.left)+self.countNodes(root.right)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



大约用时：30 分钟

LC103.二叉树的锯齿形层序遍历

bfs, https://leetcode.cn/problems/binary-tree-zigzag-level-order-traversal/

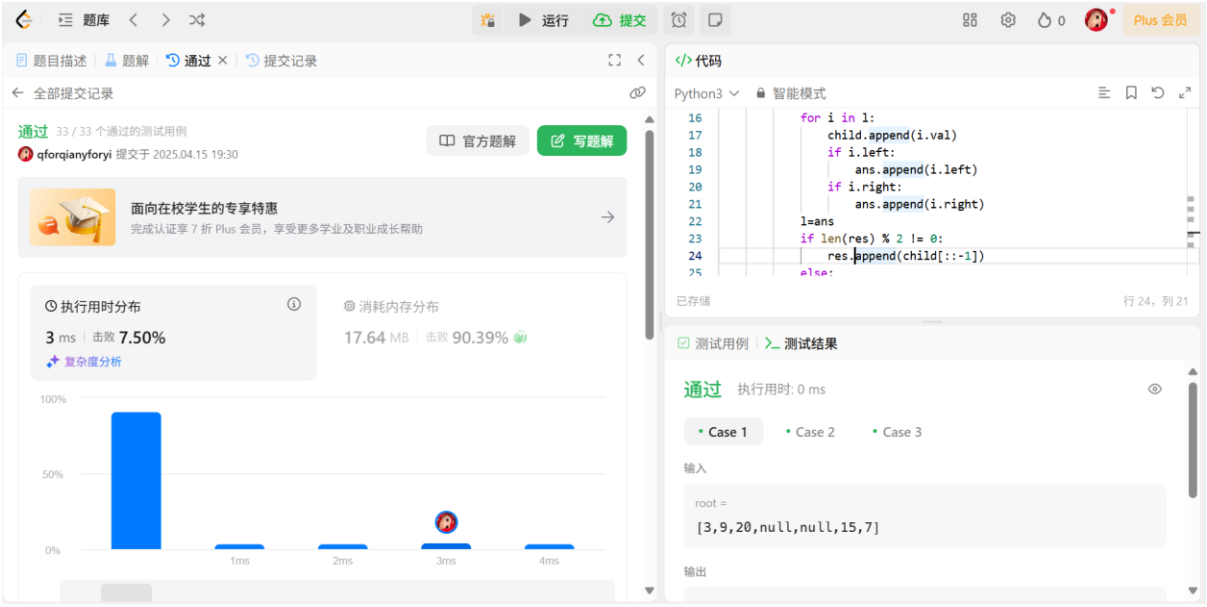
思路：bfs 遍历每一层，收集每层的节点，并遍历这个收集的节点，如果是偶数层正常存入最终列表 res 中，如果是奇数层就反过来，直到遍历完整个树，输出 res。

代码：

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def zigzagLevelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
```

```
if not root:
    return []
res=[]
l=[root]
while l:
    ans=[]
    child=[]
    for i in l:
        child.append(i.val)
        if i.left:
            ans.append(i.left)
        if i.right:
            ans.append(i.right)
    l=ans
    if len(res) % 2 != 0:
        res.append(child[::-1])
    else:
        res.append(child)
return res
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



大约用时：45 分钟

M04080:Huffman 编码树

greedy, http://cs101.openjudge.cn/practice/04080/

思路：首先用堆来让最小的值始终在最前面，当堆里有>1 个值时进行合并成新的节点，即每次取出最小的两个值并记录 cost，同时在每次循环中累加 total（所有叶子的加权路径长度之和），并将每次的 cost 推入堆中（每次循环合并最小的两项构造一棵带权路径最短的树），最后 wi 没有值时便输出 total。

代码：

```
import heapq

n=int(input())
wi=list(map(int, input().split()))

heapq.heapify(wi)
total=0

while len(wi)>1:
```

```
    s1=heapq.heappop(wi)
    s2=heapq.heappop(wi)

    cost=s1+s2
    total+=cost

    heapq.heappush(wi,cost)

print(total)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

大约用时：45 分钟

M05455: 二叉搜索树的层次遍历

http://cs101.openjudge.cn/practice/05455/

思路：先用 set 去重，再根据原始的顺序进行排序，然后构建二叉搜索树（遍历每一层，寻找合适位置来插入每个节点，即通过比较两个数的大小，如果<根节点便插入左边，如果>插入右边，每次递归插入确保是在叶子节点），再用队列来从根节点开始循

环并弹出加入 res 列表，每次检查当前节点是否有子节点，如果有就逐层加入它的左右节点（从左到右），重复直到队列中为空。

代码：

```
from collections import deque

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

def tree(root,value):
    if not root:
        return TreeNode(value)
    if value < root.val:
        root.left=tree(root.left,value)
    elif value > root.val:
        root.right=tree(root.right,value)
    return root

n=list(map(int,input().split()))
m=list(set(n))
m.sort(key=n.index)
# print(m)
root=None
for i in m:
    root=tree(root,i)

res=[]
q=deque([root])
while q:
    node=q.popleft()
    res.append(node.val)
    if node.left:
        q.append(node.left)
    if node.right:
        q.append(node.right)
```

print(' '.join(map(str,res)))

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

#48924098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,val):
        self.val=val
        self.left=None
        self.right=None

def tree(root,value):
    if not root:
        return TreeNode(value)
    if value < root.val:
        root.left=tree(root.left,value)
    elif value > root.val:
        root.right=tree(root.right,value)
    return root

n=list(map(int,input().split()))
m=list(set(n))
m.sort(key=n.index)
# print(m)
root=None
for i in m:
    root=tree(root,i)

res=[]
q=deque([root])
while q:
    node=q.popleft()
    res.append(node.val)
    if node.left:
        q.append(node.left)
    if node.right:
        q.append(node.right)

print(' '.join(map(str,res)))
```

基本信息

#: 48924098
题目: 05455
提交人: 2400093012 苏倩仪
内存: 3624kB
时间: 18ms
语言: Python3
提交时间: 2025-04-16 13:26:56

English 帮助 关于

大约用时：1 小时 30 分钟

M04078: 实现堆结构

手搓实现，http://cs101.openjudge.cn/practice/04078/

类似的题目是 晴问 9.7: 向下调整构建大顶堆，https://sunnywhy.com/sfbj/9/7

思路：用 heapq 可以很方便的实现，如果第一个元素是 1 表示加入堆，如果是 2 则输出并删除堆里最小的数

代码：

```
import heapq

n=int(input())
res=[]
for _ in range(n):
    type=list(map(int,input().split()))
    heapq.heapify(res)
    if type[0]==1:
        heapq.heappush(res,type[1])
    elif type[0]==2:
        a=heapq.heappop(res)
        print(a)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

大约用时：30 分钟

T22161: 哈夫曼编码树

greedy, http://cs101.openjudge.cn/practice/22161/

思路：需要进行类定义来进行树的节点，除了.val/.char/.left/.right，还需要定义 lt 来确保当两个输入的权重一样，则根据字母的顺序进行排序。接着定义 tree 函数，把每个字符变成节点，然后每次从堆里取出最小的两个，合并成一个新节点（权重相加），然后继续推入堆里，重复直到剩一个节点（哈夫曼树的根节点），然后定义 codes 函数进行哈夫曼编码，从根节点开始递归遍历，如果向左记 0，向右记 1，当走到叶子节点就把路径记录下来（即字符的编码），之后再对接下来的输入进行编码和解码，编码即把原始字符变成二进制编码（用之前用字典记录下的编码进行对应），解码即用给定的二进制编码在哈夫曼树里走一次得到最后的字符。

代码：

```
import heapq

class TreeNode:
    def __init__(self,val,char):
        self.val=val # 权重
        self.char=char # 节点的字符
        self.left=None
        self.right=None

    def __lt__(self,other): # 为了让节点能放进最小堆里，会根据权重排序，权重相同的就按字符字母序大小排
        if self.val == other.val: # 左子节点 < 右子节点
            return self.char < other.char # 如果权重一样，按字母顺序比
        return self.val < other.val # 否则就比较权重大小


# 构建哈夫曼树
def tree(frequent):
    # 创建最小堆（优先队列）
    heap=[]
    for char,freq in frequent.items():
        heapq.heappush(heap,TreeNode(freq,char)) # 将每个字符和频率推入堆中
```

```
    # 合并堆中的节点，直到剩下一个节点为止
    while len(heap) > 1:
        # 弹出两个权重最小的节点
        s1=heapq.heappop(heap)
        s2=heapq.heappop(heap)

        # 创建一个新的节点（权重是两个节点的和）
        merged=TreeNode(s1.val+s2.val, s1.char+s2.char)
        merged.left=s1
        merged.right=s2

        # 将合并后的节点插入堆中
        heapq.heappush(heap,merged)

    # 返回哈夫曼树的根节点
    return heap[0]


# 生成哈夫曼编码
def codes(root,current_code="",code_map={}):
    if root is None:
        return

    # 如果是叶子节点，则将该字符的编码存入字典
    if not root.left and not root.right:
        code_map[root.char]=current_code
        return

    # dfs
    codes(root.left, current_code + '0', code_map)  # 左边 0
    codes(root.right, current_code + '1', code_map)  # 右边 1

    return code_map

n=int(input())
frequent={}

for _ in range(n):
    char,freq=input().split()
```

```
    freq=int(freq)
    frequent[char]=freq

root=tree(frequent)
code_map=codes(root)

while True:
    try:
        # 输入一行编码或解码的请求
        i=input()
        if i.isalpha(): # 如果输入的是字母串，进行编码
            print(''.join([code_map[char] for char in i]))
        else: # 如果输入的是 01 串，进行解码
            decode=[]
            cur=root
            for bit in i:
                if bit == '0':
                    cur = cur.left
                else:
                    cur = cur.right
                # 如果到达叶子节点，就添加字符并回到根节点
                if not cur.left and not cur.right:
                    decode.append(cur.char)
                    cur = root

            print(''.join(decode))
    except EOFError:
        break
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

#48930576提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 48930576
题目: 22161
提交人: 2400093012 另倩仪
内存: 3676kB
时间: 19ms
语言: Python3
提交时间: 2025-04-16 19:15:31

源代码

```
import heapq

class TreeNode:
    def __init__(self,val,char):
        self.val=val # 权重
        self.char=char # 节点的字符
        self.left=None
        self.right=None

    def __lt__(self,other): # 为了让节点能放进最小堆里，会根据权重排序，权重相同
        if self.val == other.val: # 左子节点 < 右子节点
            return self.char < other.char # 如果权重一样，按字母顺序比
        return self.val < other.val # 否则就比较权重大小


# 构建哈夫曼树
def tree(frequent):
    # 创建最小堆（优先队列）
    heap=[]
    for char,freq in frequent.items():
        heapq.heappush(heap,TreeNode(freq,char))  # 将每个字符和频率插入堆中

    # 合并堆中的节点，直到剩下一个节点为止
    while len(heap) > 1:
        # 弹出两个权重最小的节点
        x1=heapq.heappop(heap)
        x2=heapq.heappop(heap)

        # 创建一个新的节点（权重是两个节点的和）
        merged=TreeNode(x1.val+x2.val, x1.char+x2.char)
        merged.left=x1
        merged.right=x2

        # 将合并后的节点插入堆中
        heapq.heappush(heap,merged)

    # 返回哈夫曼树的根节点
    return heap[0]


# 生成哈夫曼编码
def codes(root,current_code="",code_map={}):
    if root is None:
        return

    # 如果是叶子节点，则将该字符的编码存入字典
    if not root.left and not root.right:
        code_map[root.char]=current_code
        return

    # dfs
    codes(root.left, current_code + '0', code_map)  # 左边0
    codes(root.right, current_code + '1', code_map)  # 右边1

    return code_map

n=int(input())
frequent={}

for _ in range(n):
    char,freq=input().split()
    freq=int(freq)
    frequent[char]=freq

root=tree(frequent)
code_map=codes(root)

while True:
    try:
        # 输入一行编码或解码的请求
        i=input()
        if i.isalpha():  # 如果输入的是字母串，进行编码
            print(''.join([code_map[char] for char in i]))
        else:  # 如果输入的是01串，进行解码
            decode=[]
            cur=root
            for bit in i:
                if bit == '0':
                    cur = cur.left
                else:
                    cur = cur.right
                # 如果到达叶子节点，就添加字符并回到根节点
                if not cur.left and not cur.right:
                    decode.append(cur.char)
                    cur = root

            print(''.join(decode))
    except EOFError:
        break
```

大约用时：2 小时

2. 学习总结和收获

<mark>如果发现作业题目相对简单，有否寻找额外的练习题目，如“数算 2025spring 每日选做”、LeetCode、Codeforces、洛谷等网站上的题目。</mark>

这次很多题目看不明白…需要给 ai 解释之后才看得懂…前两题还可以原理挺简单的（虽然写的时候还是犯了一些小错误）~第三题一开始看不懂但是给 ai 翻译之后其实也很简单，也突然顿悟 heapq 的使用场景了，比起之前在计概需要自己排序+添加元素真的方便好多。第四题真的看不懂题目，问了 ai 也明白了，感觉层序遍历的感觉也好像这里的递归插入操作，都是一层一层往下找到叶子再一步步构建/输出整个树，稍微消除一点我对树感到抽象的感觉了 hhh。第五题感觉是迟来的基础题 TvT，第一次看到 heapq 还是之前某次作业的最后一道难题（当时都傻了），后面一头雾水地又做了几次 heapq，现在看到这样的一题可以很轻易的做出来，还蛮开心的哈哈哈。第六题好难…虽然题目看得懂，大概的思路也有，但是写起来还是很难，最后看了题解花了至少一小时半才看懂 hhh，问了 ai 才知道还需要另外定义一个判断当权重相同则按照字母顺序排序的类定义，一直都对类定义不明白，只有今年的链表和树稍微接触了，但还是只有最浅的理解 TT。