Assignment #9: Huffman, BST & Heap

Updated 1834 GMT+8 Apr 15, 2025

2025 spring, Complied by <mark>同学的姓名、院系</mark>

姓名：李彦臻
学号：2300010821
学院：数学科学学院

> **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

LC222.完全二叉树的节点个数

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

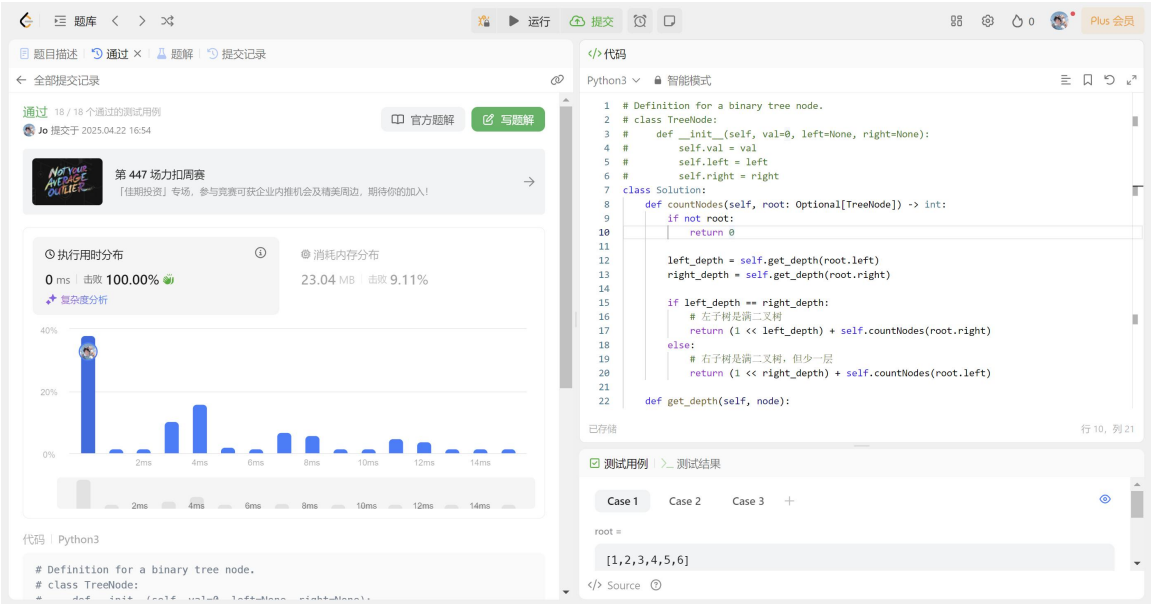
思路：对于完全二叉树，可以通过一直向左遍历和一直向右遍历来得到左右子树的高度。先判断是否为满二叉树：如果左子树的高度等于右子树的高度，则说明该树是满二叉树，其节点总数为 2^h（树的高度）- 1；反之，如果左右子树高度不相等，则递归计算左子树和右子树的节点数，然后加 1（对应当前节点）。
用时：15mins

代码：

```python

```

```
class Solution:
    def countNodes(self, root: Optional[TreeNode]) -> int:
        if not root:
            return 0

        left_depth = self.get_depth(root.left)
        right_depth = self.get_depth(root.right)

        if left_depth == right_depth:
            # 左子树是满二叉树
            return (1 << left_depth) + self.countNodes(root.right)
        else:
            # 右子树是满二叉树，但少一层
            return (1 << right_depth) + self.countNodes(root.left)

    def get_depth(self, node):
        depth = 0
        while node:
            depth += 1
            node = node.left
        return depth
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



LC103.二叉树的锯齿形层序遍历

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

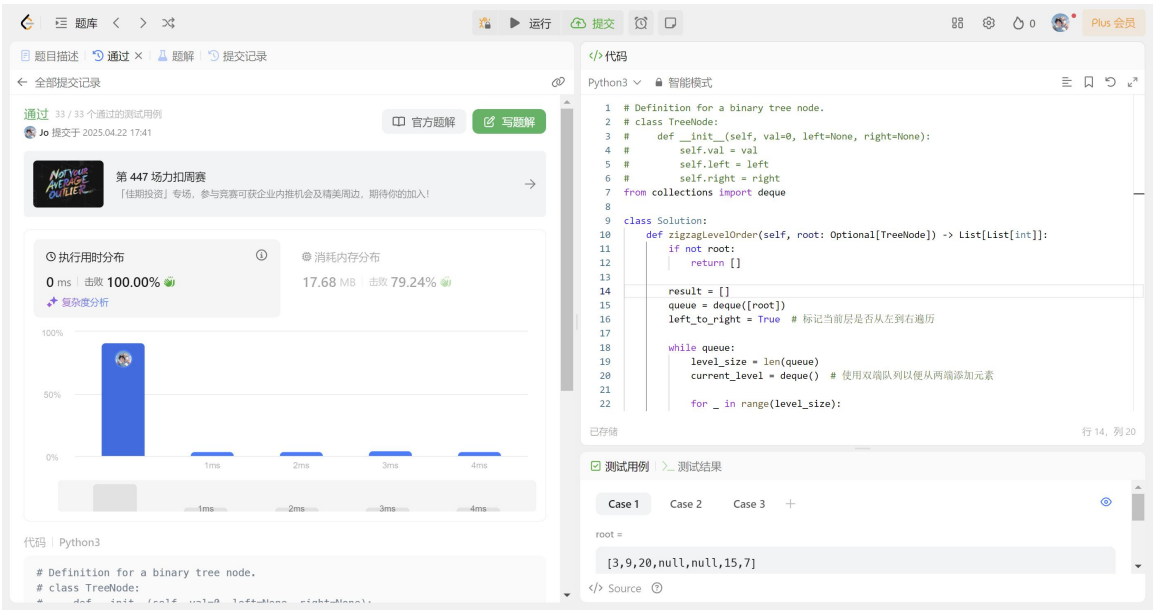
思路：使用 BFS 方法即可：先利用队列来层次遍历二叉树（每次处理一层）；接着，确定遍历方向：使用一个标志位来指示当前层是从左到右还是从右到左遍历。初始时，第一层从左到右遍历，下一层则相反，依此类推；最后，再根据当前层的方向，将节点值按顺序或逆序添加到结果列表中。
用时：20mins

代码：

```python
```
```
from collections import deque

class Solution:
    def zigzagLevelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        if not root:
            return []

        result = []
        queue = deque([root])
        left_to_right = True  # 标记当前层是否从左到右遍历

        while queue:
            level_size = len(queue)
            current_level = deque()  # 使用双端队列以便从两端添加元素

            for _ in range(level_size):
                node = queue.popleft()

                if left_to_right:
                    current_level.append(node.val)
                else:
                    current_level.appendleft(node.val)

                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
```

```
            result.append(list(current_level))
            left_to_right = not left_to_right  # 切换方向

        return result
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



M04080:Huffman 编码树

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

思路：用解决赫夫曼问题的方法来做即可：先将所有权值看作一个个独立的节点，放入优先队列，每次取出权值最小的两个节点合并，合并代价为两者权值之和；接着，把合并后的新节点重新放入队列中；重复此过程，直到只剩一个根节点。
最后，计算所有合并过程的代价之和，即为最小外部带权路径长度总和。
用时：20mins

代码：

```python

```

```
import heapq
```

```
def min_weighted_path_length(n, weights):
    heapq.heapify(weights)  # 构建最小堆
    total_cost = 0

    while len(weights) > 1:
        a = heapq.heappop(weights)
        b = heapq.heappop(weights)
        merged = a + b
        total_cost += merged
        heapq.heappush(weights, merged)

    return total_cost

n = int(input())
weights = list(map(int, input().split()))print(min_weighted_path_length(n, weights))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

M05455: 二叉搜索树的层次遍历

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

思路：先使用 set 去重，并在构建 BST 时，如果小于当前节点值，插入左子树，反之亦然。层次遍历使用 queue 实现即可~
用时：15mins

代码：

```python

```
```
from collections import deque

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

def insert(root, val):
    if root is None:
        return TreeNode(val)
    if val < root.val:
        root.left = insert(root.left, val)
    elif val > root.val:
        root.right = insert(root.right, val)
    return root

def level_order_traversal(root):
    if not root:
        return []
    result = []
    queue = deque([root])
    while queue:
        node = queue.popleft()
        result.append(node.val)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return result

values = list(map(int, input().split()))
unique_values = sorted(set(values), key=values.index)  # 去重

# 构建 BST
root = None
for val in unique_values:
    root = insert(root, val)
```

```
result = level_order_traversal(root)
print(' '.join(map(str, result)))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

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：

代码：

```python

```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

T22161: 哈夫曼编码树

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

思路：

代码：

```python

```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

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>

通过这次作业学习到了不少知识，比如 huffman 问题的解决方法：

赫夫曼算法核心步骤：

1. 将所有权值看作一个个独立的节点，放入优先队列（小根堆）。
2. 每次取出权值最小的两个节点合并，合并代价为两者权值之和。
3. 把合并后的新节点重新放入队列中。
4. 重复此过程，直到只剩一个根节点。
5. 计算所有合并过程的代价之和，即为最小外部带权路径长度总和。

以及完全二叉树的一些性质：

5. 高效计算节点数

- **利用高度差快速计算**：可以通过比较左右子树的高度来判断子树是否为满二叉树：
  - 如果左子树的高度等于右子树的高度，则左子树是满二叉树，节点数为 $2^{\text{left\_depth}} - 1$，加上根节点和递归计算右子树的节点数。
  - 如果高度不等，则右子树是满二叉树（高度比左子树少 1），节点数为 $2^{\text{right\_depth}} - 1$，加上根节点和递归计算左子树的节点数。
- **时间复杂度**：由于每次递归都能将问题规模减半，时间复杂度为 $O(\log n \times \log n)$。

6. 应用场景

- **堆的实现**：完全二叉树是二叉堆（如最大堆、最小堆）的基础结构，支持高效的插入、删除和取最值操作。
- **优先队列**：因其高效的特性，常用于实现优先队列。
- **空间优化**：用数组存储完全二叉树可以节省指针开销，适合内存受限的环境。

总的来讲，感觉我期中那一段时间落下的知识还没完全补回来，还得多加练习~