入选理由

1. 和哈希以及树专题都有联动，帮助大家复习哈希表。

标签

- 哈希表
- 树
- BFS

难度

- 中等

题目地址（Top-View-of-a-Tree）

https://binarysearch.com/problems/Top-View-of-a-Tree

题目描述

Given a binary tree root, return the top view of the tree, sorted left-to-right.

Constraints

n ≤ 100,000 where n is the number of nodes in root

前置知识

- BFS
- 哈希表

思路

首先我们分析一下题目。

1. 题目中给的示例并没有输出树中所有的节点，可以看到节点 4 和 节点 5 并没有被输出。其原因在于这两个节点被挡住了。具体来说是节点 4 被节点 1 挡住了，节点 5 被节点 3 挡住了。

2. 同时，题目要求我们返回的顺序是从左到右的。如何做到这一点?

对于第一个问题，我们可以记录一下每一个节点的横坐标和纵坐标。如果两个节点横坐标相同，那么纵坐标小的覆盖纵坐标大的。当然这需要我们使用层次遍历，这样就可以实现纵坐标覆盖了。

> 有的同学有疑问，这样不是纵坐标大的覆盖纵坐标小的么？实际上，我们都可以做到，只是代码细节会有一点点不同。比如要实现纵坐标大的覆盖小的，只需要去掉这行代码：if pos not in d:

对于第二个问题，我们需要记录横坐标，最终按照横坐标从小到大的顺序输出即可。

因此解决问题的核心在于记录横纵坐标。 假设一个节点的坐标为 (x, y)。这样进行遍历的时候左节点就是 (x - 1, y + 1)，右节点就是 (x + 1, y + 1)。我们只需要初始化 root 为 (0,0) 然后遍历，遍历过程中将所有节点的横纵坐标以及 value 放到一个哈希表中，哈希表的 key 是横坐标，value 是节点值，最终将哈希值中的数据排序输出即可。

> 当然你给左右子节点编号是 (x-dx, y+dy), (x+dx, y+dy) 也都是可以的，其中 dx 和 dy 为任意正整数，不过这并没有必要。

关键点

- 对节点进行横纵坐标的编号，以及节点和左右子节点的编号关系。
- 层次遍历简化纵坐标的判断

代码

代码支持 Python3:

```
# class Tree:
#     def __init__(self, val, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def solve(self, root):
        q = collections.deque([(root, 0)])
        d = {}
        while q:
            cur, pos = q.popleft()
            if pos not in d:
                d[pos] = cur.val
```

```
            if cur.left:
                q.append((cur.left, pos - 1))
            if cur.right:
                q.append((cur.right, pos + 1))
        return list(map(lambda x:x[1], sorted(d.items(),key=lambda x: x[0])))
```

令 n 节点数

复杂度分析

- 时间复杂度：我们使用了排序，并且所有节点最多只处理一次，因此时间复杂度为 $O(nlogn)$。
- 空间复杂度：哈希表最多容纳 n 个元素，因此空间复杂度为 $O(n)$

上一页 下一页