Daily Coding Problem #215

Problem

This problem was asked by Yelp.

The horizontal distance of a binary tree node describes how far left or right the node will be when the tree is printed out.

More rigorously, we can define it as follows:

- The horizontal distance of the root is 0.
- The horizontal distance of a left child is `hd(parent) - 1`.
- The horizontal distance of a right child is `hd(parent) + 1`.

For example, for the following tree, `hd(1) = -2`, and `hd(6) = 0`.

```
             5
          /     \
        3         7
      /  \      /   \
    1      4  6       9
```

```
  /                 /
 0                 8
```

The bottom view of a tree, then, consists of the lowest node at each horizontal distance. If there are two nodes at the same depth and horizontal distance, either is acceptable.

For this tree, for example, the bottom view could be `[0, 1, 3, 6, 8, 9]`.

Given the root to a binary tree, return its bottom view.

Solution

The key here is to create a hash table mapping each horizontal distance of the tree to a value at that distance. If we find a new node with the same horizontal distance but a greater depth, we should replace the value with the new node. This data structure could have the following format: `{hd: (value, depth)}`.

How can we populate this table? We can choose any tree traversal method- here we will use pre-order. Whenever we reach a new horizontal distance, we populate our table with the value and depth of the node we are currently looking at. If we return to that distance with a new node, we check to see if we are at a deeper level, and if so, replace the value and depth.

One final issue is that Python dictionaries are not ordered, so the keys to our table may not be sorted by increasing horizontal distance. To avoid having to sort our dictionary, we can find the min and max key and loop over the values in between.

The full code would look like this:

```python
def traverse(root, hd_table, distance=0, level=0):

    if not root:
        return []

    if distance not in hd_table or hd_table[distance][1] <= level:
        hd_table[distance] = (root.val, level)
    traverse(root.left, hd_table, distance - 1, level + 1)
    traverse(root.right, hd_table, distance + 1, level + 1)

    return hd_table

def get_bottom_view(root):
    hd_table = traverse(root, {})

    min_hd, max_hd = min(hd_table), max(hd_table)
    return [hd_table[k][0] for k in range(min_hd, max_hd + 1)]
```

This algorithm will take `O(N)` time to traverse the tree and print out the keys, and use potentially `O(N)` space to store the values for all the horizontal distances.