Problems On Trees

Created	@September 7, 2022 8:35 PM
Class	
Type	
Materials	
Reviewed	☐

Problem 1:

Given a binary tree, print the level order traversal of the binary tree level wise.

```
/**
 *
 *              10
 *            /   \
 *          5      15
 *         /      / \
 *        1      24  33
 *       /
 *      0
*/
```

So for the above binary tree, normal level order will be, 10, 5, 15, 1, 23, 33, 0

But now instead of printing the whole level order together, we want to print every level in a new line. So level order level wise for the above tree will look like

```
10
5 15
1 24 33
0
```

Solution:

If we try to carefully debug our original level order approach, then we can observe something.

When all the elements of the 0th level got removed then only the elements of the 1st level were present in the queue. When all the elements of the 1st level got removed then only the elements of the 2nd level were present in the queue. And So on…

So we can say, as soon as all the elements of the previous level gets removed we can observe that in our queue, all the elements of the next level is present.

Can we use this idea ? If somehow we can mark when the previous level ended, we can say that all the elements in the queue are of next level.

And for the 0th level, we know there will be always 1 element, so what we can do is, instead of just pushing the root, in the queue at first, push the root as well as a null value. Whenever we will get a null in the front of the queue, we can say all the prev level elements are gone and only next level elements are present.

```
/**
 *
 *              10
 *            /   \
 *          5      15
 *         /      / \
 *        1     24  33
 *       /
 *      0
*/

qu -> []
10
5 15
1 24 33
0
```

Problem

Binary Tree Right Side View - LeetCode

Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom. Example 1: Input: root =

https://leetcode.com/problems/binary-tree-right-side-view/

One solution can be using level order level wise so that we can detect the rightmost element of the previous level, which is the element after which we get a null.

There can be one more approach for this problem.

Can we think in terms of reverse pre order ?

Normal Preorder: Root LeftSubtree RightSubtree

Reverse preorder means : Root RightSubtree LeftSubtree, we actually reverse the order in which subtrees are read, that is right first then left

Reverse preorder is not post order

```
/**
 *
 *              10
 *            /   \
 *          5      15
 *         /      / \
 *        1      24  33
 *       /
 *      0
*/
10, 15, 33, 24, 5, 1, 0 -> Reverse pre order
```

Now in the reverse pre order, if we carefully analyse then the first time we hit a new level of the tree, we hit the rightmost node of that level first. And these set of rightmost nodes are our answer.

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
```

```
/**
 * @param {TreeNode} root
 * @return {number[]}
 */
let maxLevel = -1; // max level stores the maximum level seen so far
let result = [];
function rightView(root, level) {
  if(root == null) return;
  if(level > maxLevel) {
    // this is the first time we are hitting this level
    maxLevel = level;
    result.push(root.val);
  }
  rightView(root.right, level + 1);
  rightView(root.left, level + 1);
}

var rightSideView = function(root) {
    result = [];
    maxLevel = -1;
    rightView(root, 0);
    return result;
};
```

Problem:

Given a preorder of a binary tree such that every absent child is represented by -1, is given to us, can we form the whole binary tree ?

```
/**
 *
 *              10
 *            /   \
 *          5      15
 *         /      / \
 *        1     24  33
 *       /
 *      0
*/
Input: 10, 5, 1, 0, -1, -1, -1, -1, 15, 24, -1, -1, 33, -1, -1
```

We know that if pre order is given and we will start to read it, we will be reading a root first, then its lst, then its rst.

So what we can do is we can start reading the input and we can build a recursive logic. The current value that we are reading we will make it a root, and then in the remaining input we will build the left subtree till the time we hit -1, the moment we hit -1, we will return a null and then start building right subtree.

```
```