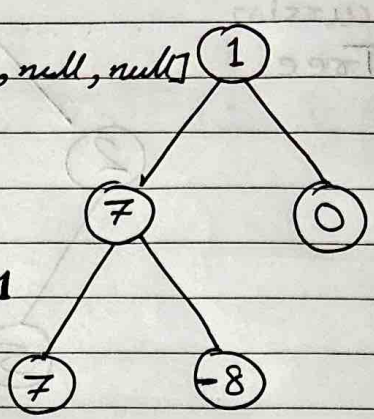
DATE

LEETCODE | 1161 |

Maximum Level Sum of a Binary Tree

Input: root = [1, 7, 0, 7, -8, null, null]



Output: 2

Note → Root level starts with 1

Approach 1 BFS

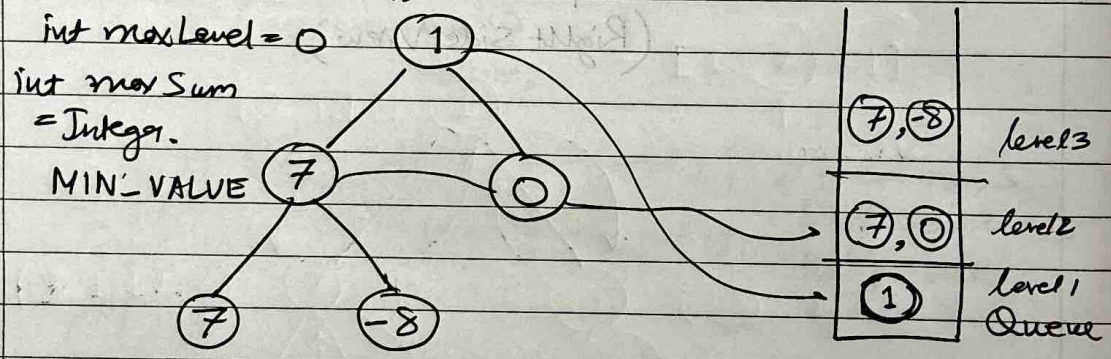
1) Base Case if (root == null) return 0;

2) Create a Queue & offer root node

int currentLevel = 1;

int maxLevel = 0

int maxSum = Integer.MIN_VALUE



3) While loop till queue is not Empty

↳ initialize. currentSum = 0 & size = queue.size()

4) Run a for loop from i=0 to size

```
for ( int i=0; i< size; i++){
    TreeNode node=queue. poll();
    currentSum += node.val;
    if (node.left != null)
        queue.offer(node.left)
    if (node.right != null)
        queue.offer(node.right)
}
```

classmate

PAGE

DATE

```
if (currentSum > maxSum) {
        maxSum = currentSum;
        maxLevel = currentLevel;
}
```

currentLevel++; ← increase levels after words for next queue position / level

5) Return maxLevel;

Approach 2 DFS

1) In the main function, create a 'List<Integer> sumLevels = new ArrayList<Integer>();

2) Create a recursive fn and call it here.

3) Recursive Function looks like below

```
void dFSTree(TreeNode node, int level, List<Integer>
                                        sumLevels) {
    // base case
    if (node == null)
        return;

    if (sumLevels.size() == level) {
        sumLevel.add(0); // initialize sum at
                         level = 0
    }
    sumLevels.set(level, sumLevel.get(level) + node.val);
```

Primary Responsibility of Recursion ↑ adds node's value to the list at index = level.

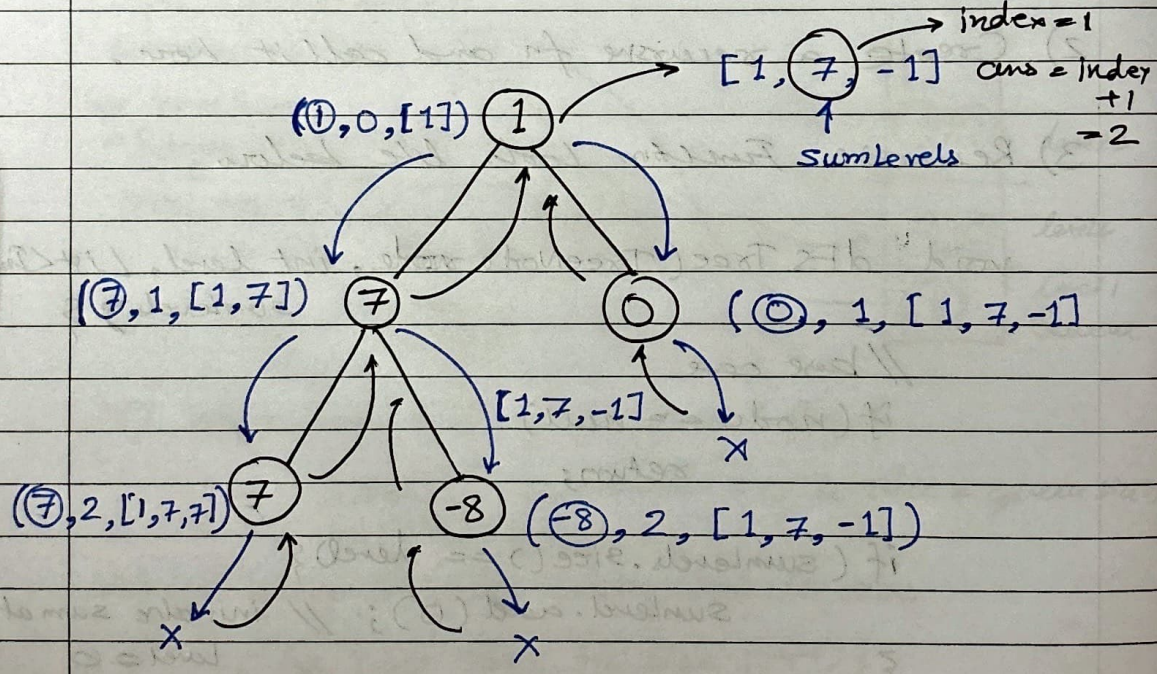
```
    // Recursive Calls here
```

```
dFSTree(node.left, level+1, sumlevels);
dFsTree(node.right, level+1, sumlevels);
}
```

4) Once Recursive calls are done, we get all values in sumlevels arraylist.

5) Iterate over sumlevels & find the index at which value is greatest

6) Return index + 1 as answer.



Recursion Tree