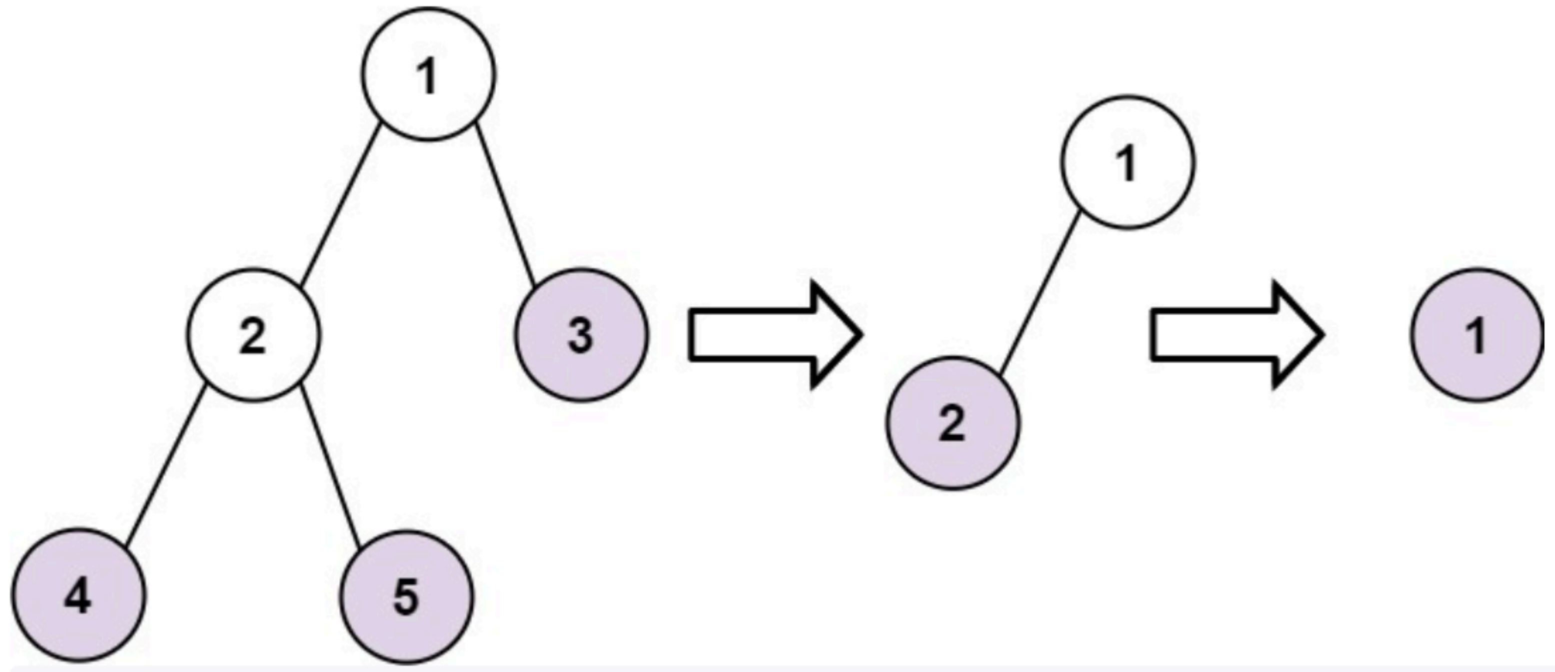
366. Find Leaves of Binary Tree

Given the `root` of a binary tree, collect a tree's nodes as if you were doing this:

- Collect all the leaf nodes.
- Remove all the leaf nodes.
- Repeat until the tree is empty.

Example 1:



```
Input: root = [1,2,3,4,5]
Output: [[4,5,3],[2],[1]]
Explanation:
[[3,5,4],[2],[1]] and [[3,4,5],[2],[1]] are also considered
correct answers since per each level it does not matter the
order on which elements are returned.
```

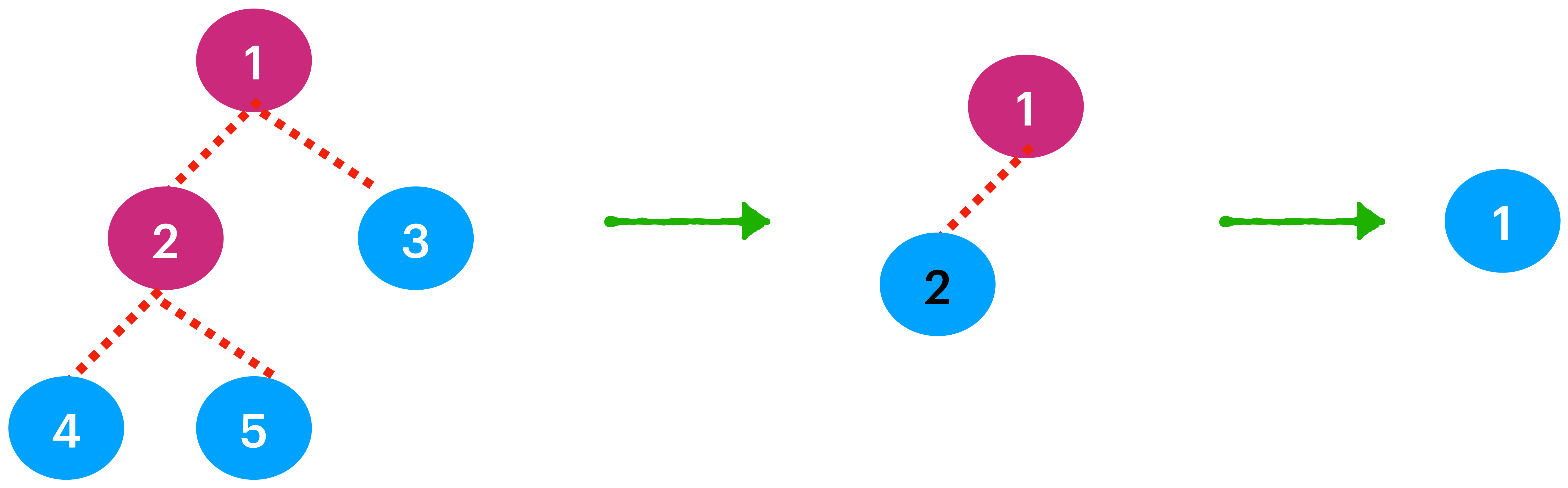
Example 2:

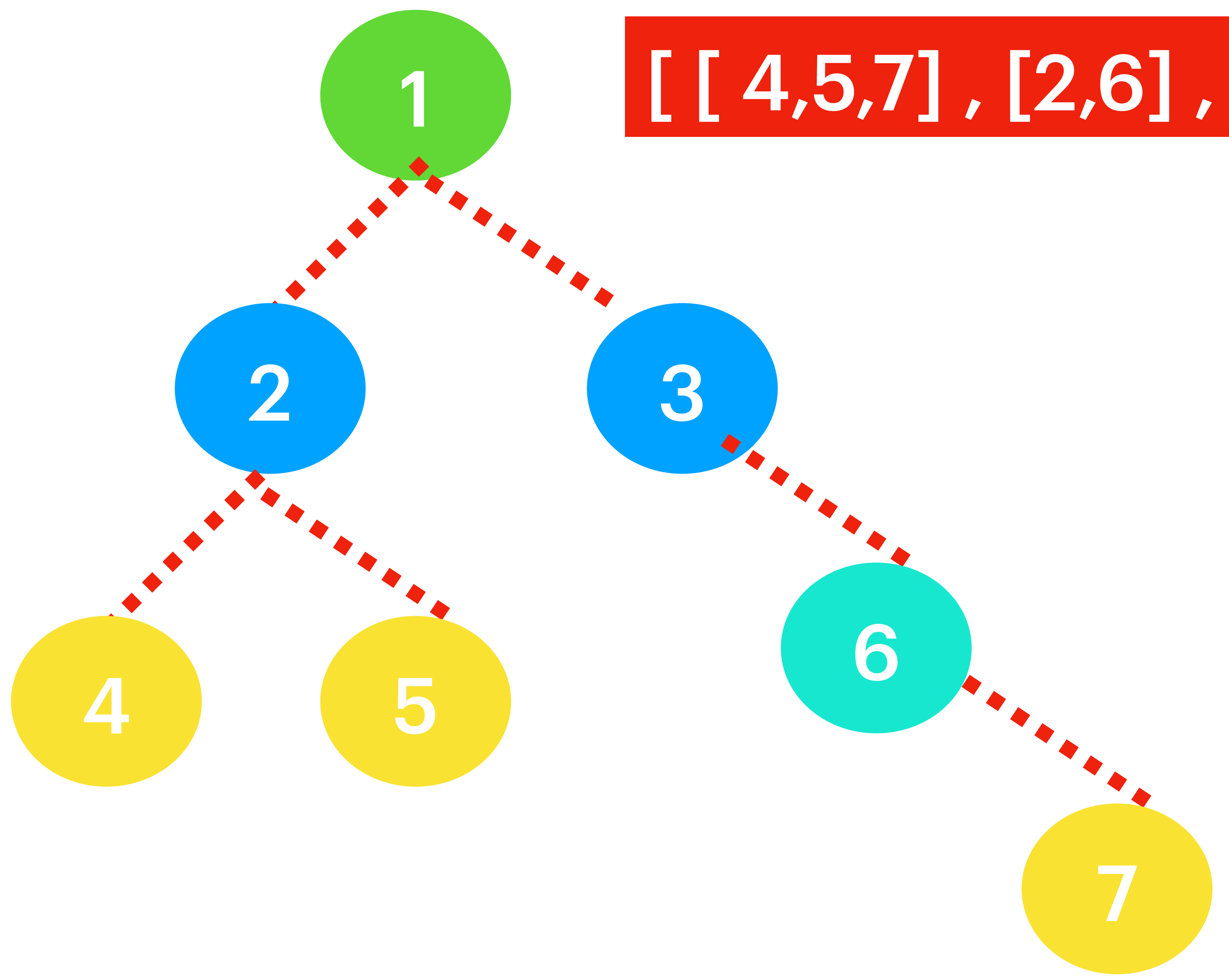
```
Input: root = [1]
Output: [[1]]
```

Constraints:

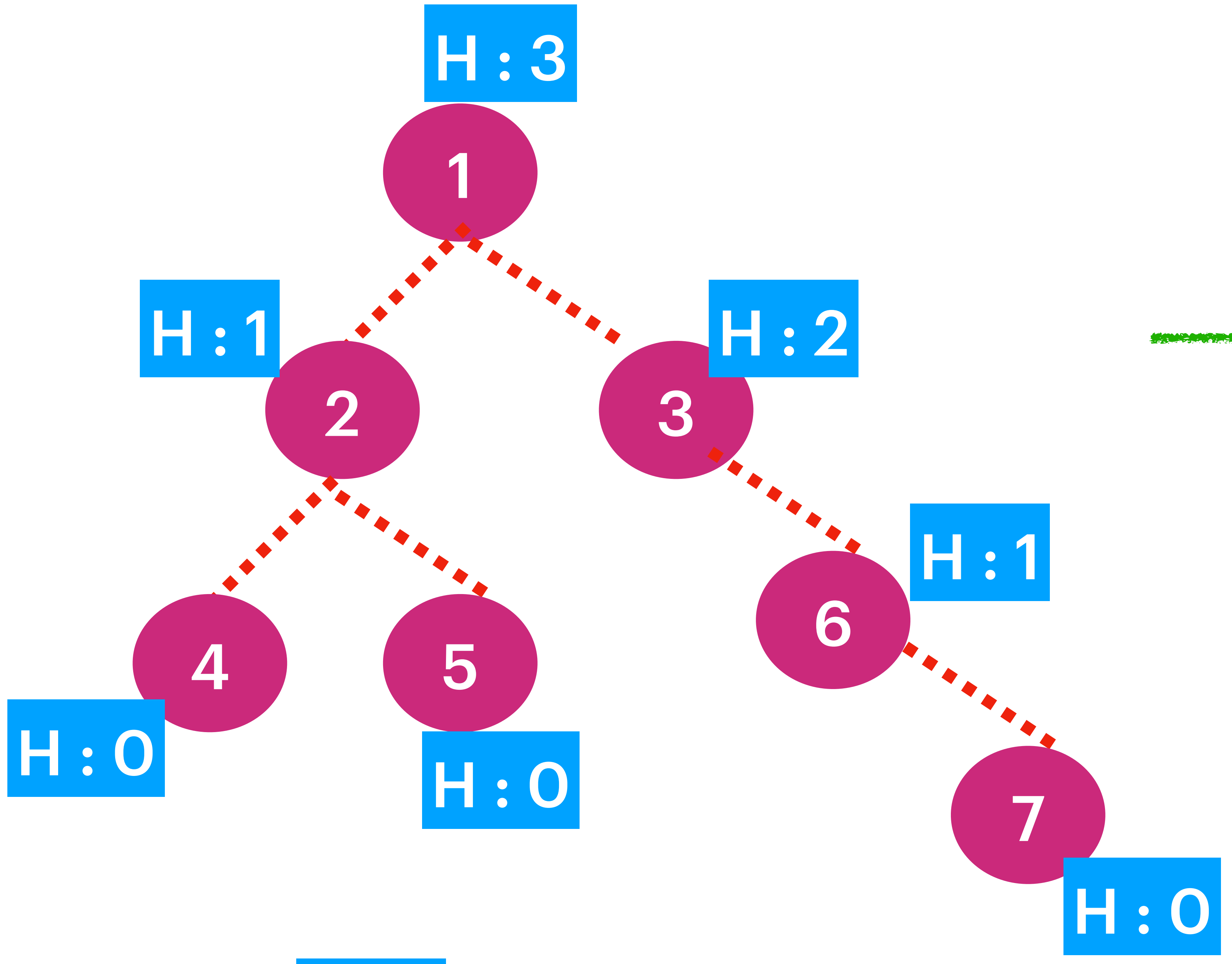
- The number of nodes in the tree is in the range `[1, 100]`.
- `-100 <= Node.val <= 100`

[[4,5,3] , [2] , [1]]





[[4,5,7] , [2,6] , [3] , [1]]

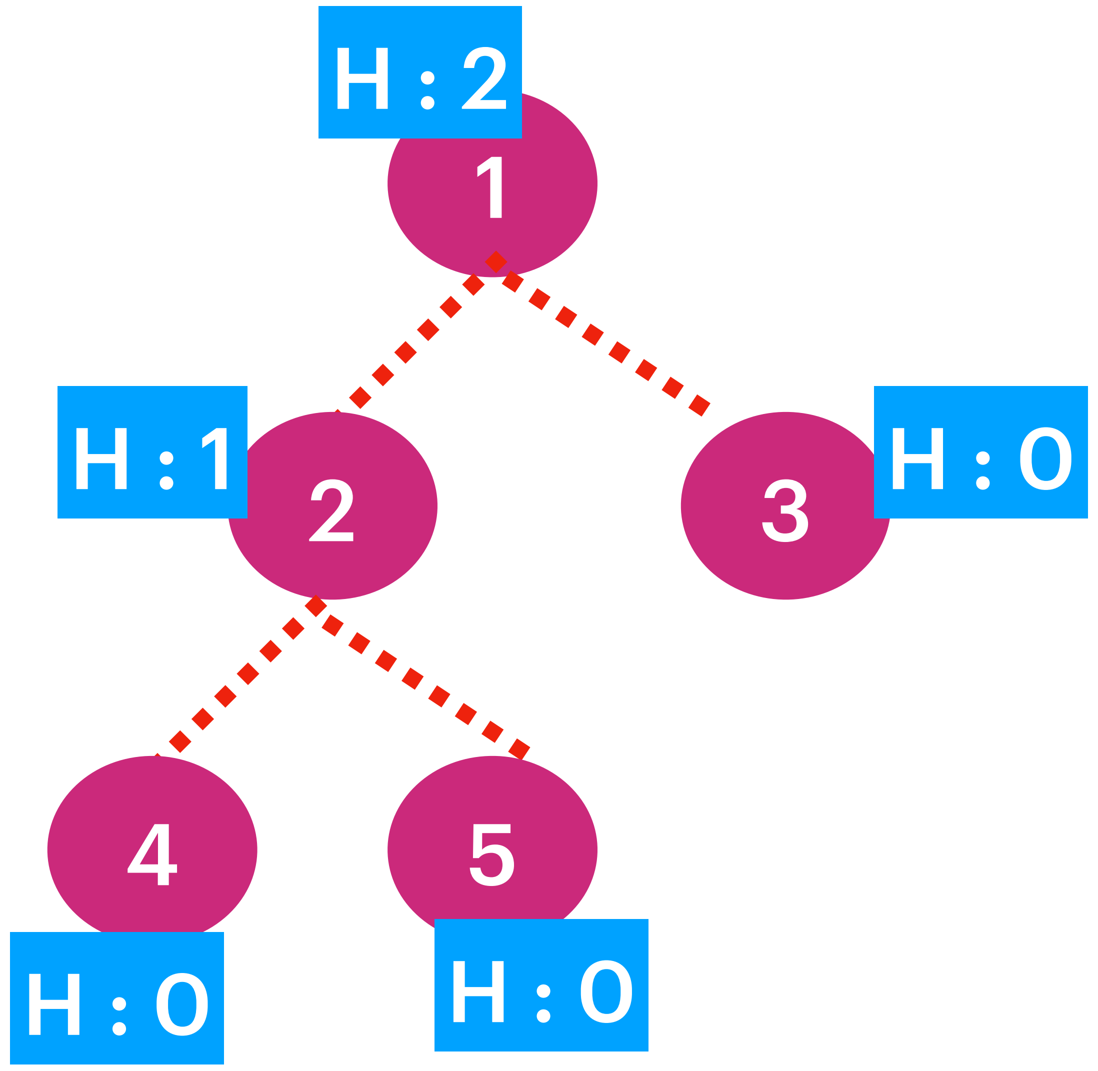


[[4,5,7] , [2,6] , [3], [1]]

Algorithm :
:: Find out height of each node
:: Group element based on height

Time Complexity: O(n)

Space Complexity: O(n)



[[4,5,3] , [2] , [1]]

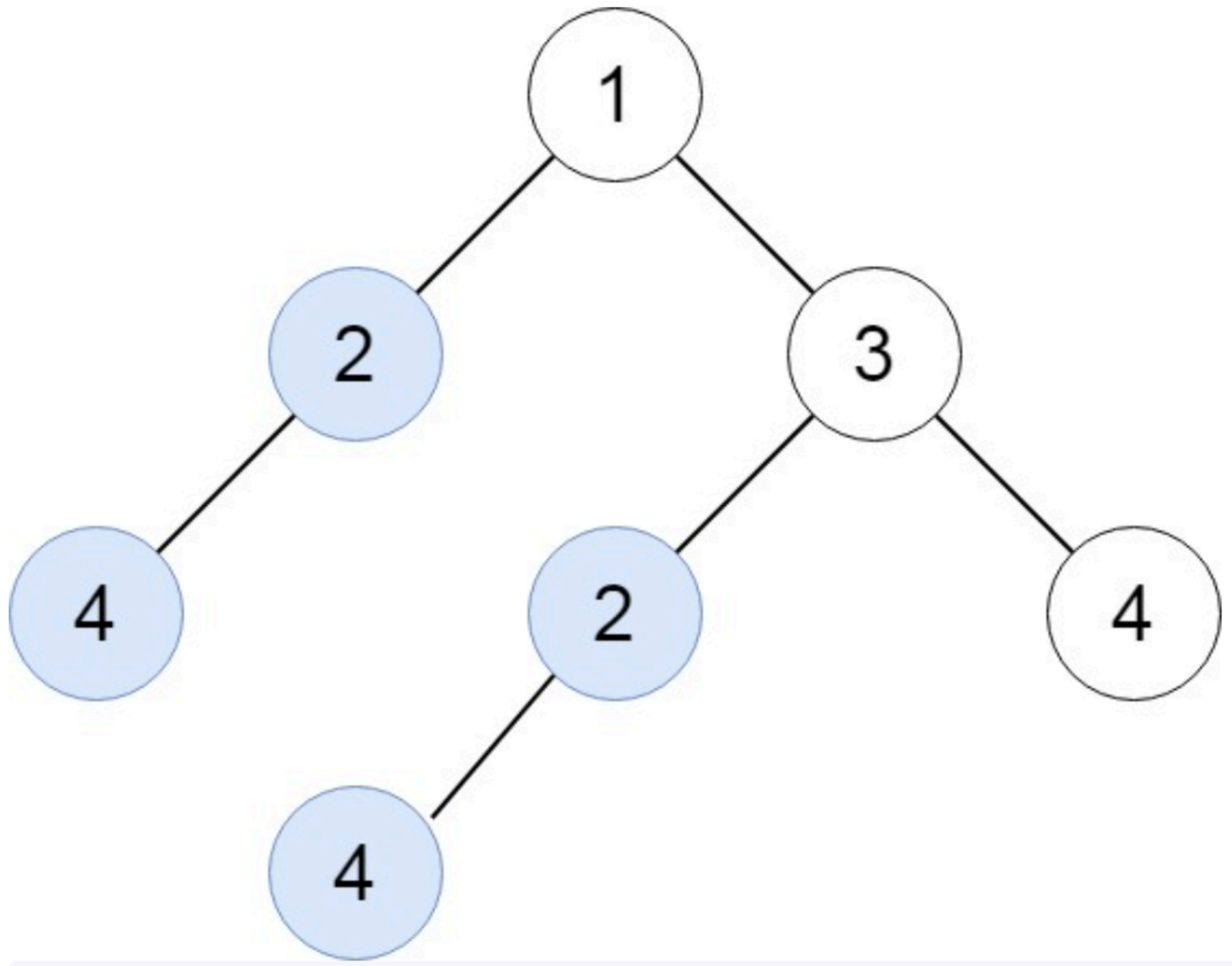
652. Find Duplicate Subtrees

Given the `root` of a binary tree, return all **duplicate subtrees**.

For each kind of duplicate subtrees, you only need to return the root node of any **one** of them.

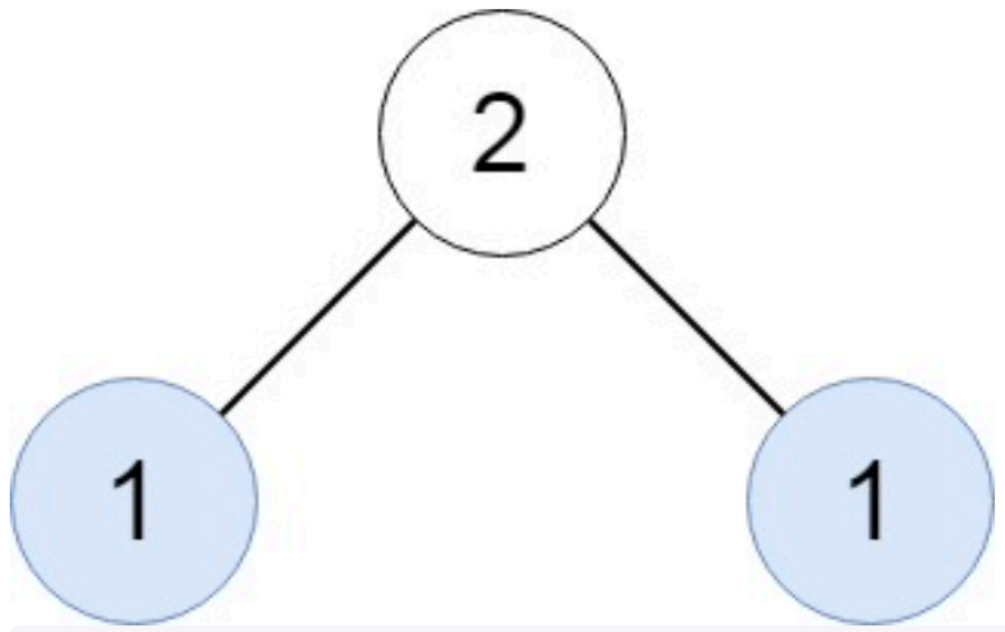
Two trees are **duplicate** if they have the **same structure** with the **same node values**.

Example 1:



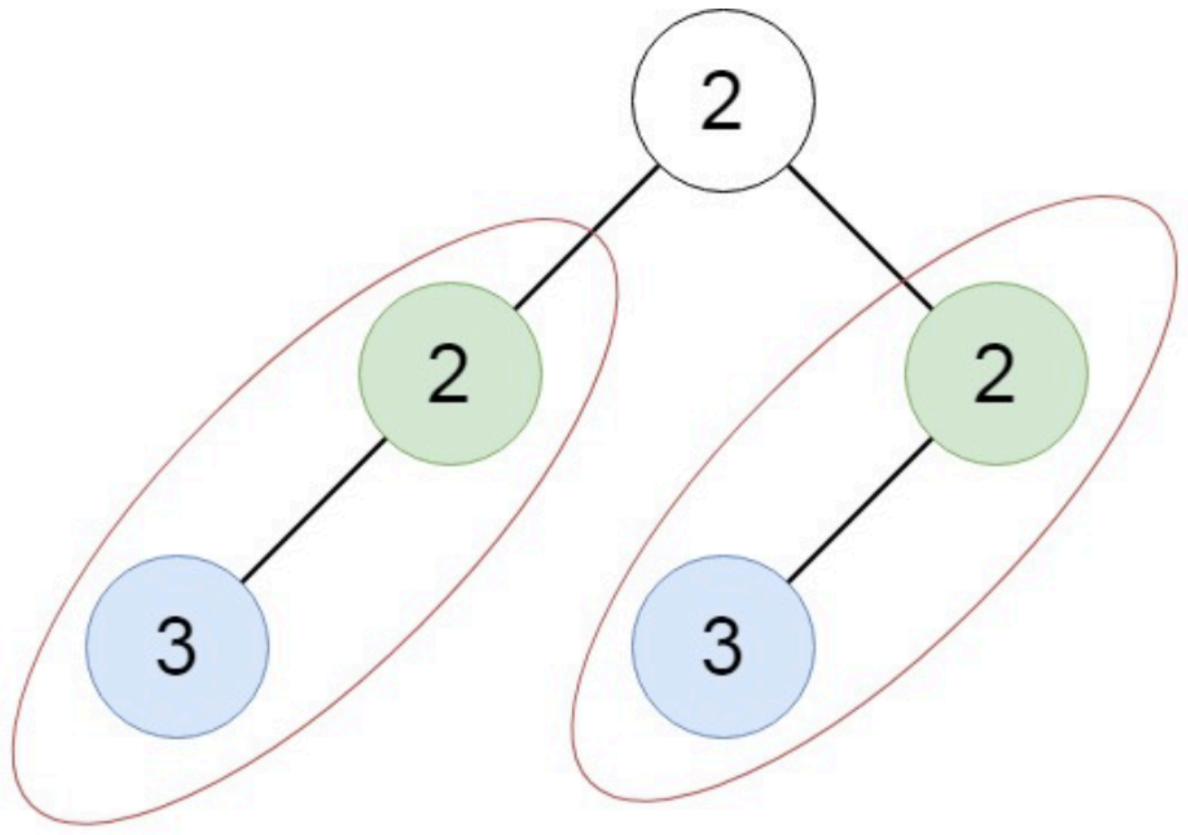
```
Input: root = [1,2,3,4,null,2,4,null,null,4]
Output: [[2,4],[4]]
```

Example 2:



```
Input: root = [2,1,1]
Output: [[1]]
```

Example 3:



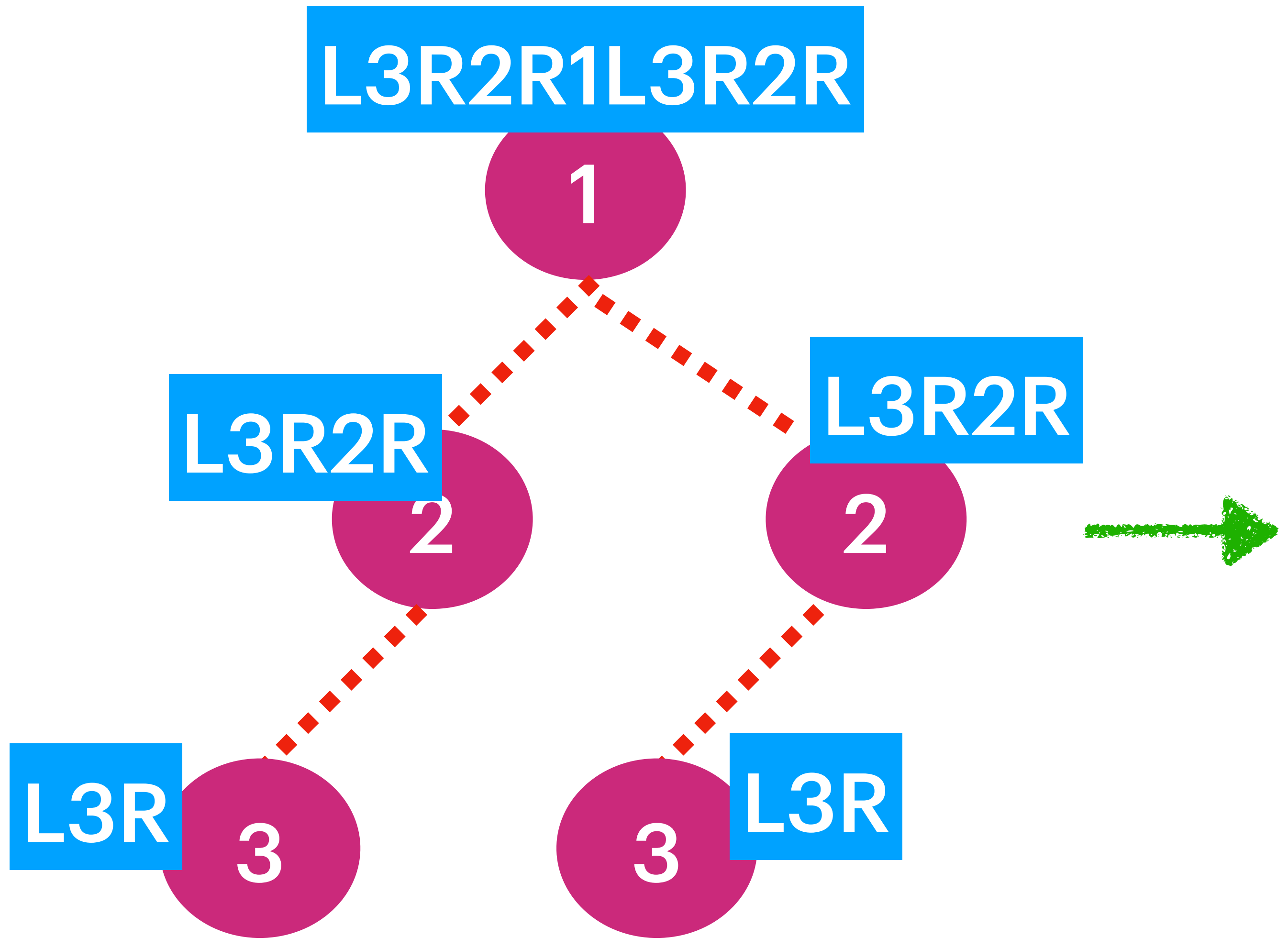
```
Input: root = [2,2,2,3,null,3,null]
Output: [[2,3],[3]]
```

Constraints:

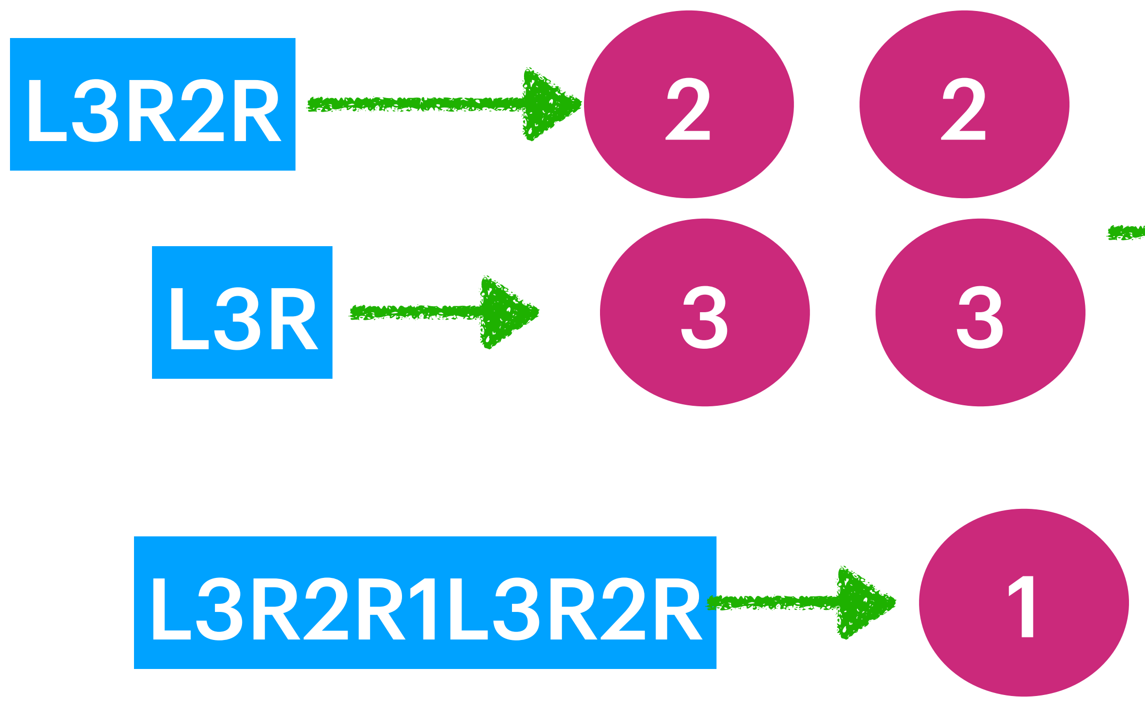
- The number of the nodes in the tree will be in the range `[1, 10^4]`
- `-200 <= Node.val <= 200`

InOrder Traversal (left-root-right):

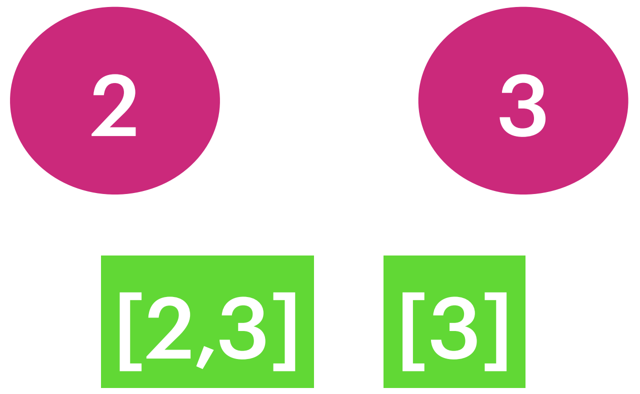
Two trees are duplicate if they
have same structure with same node values:

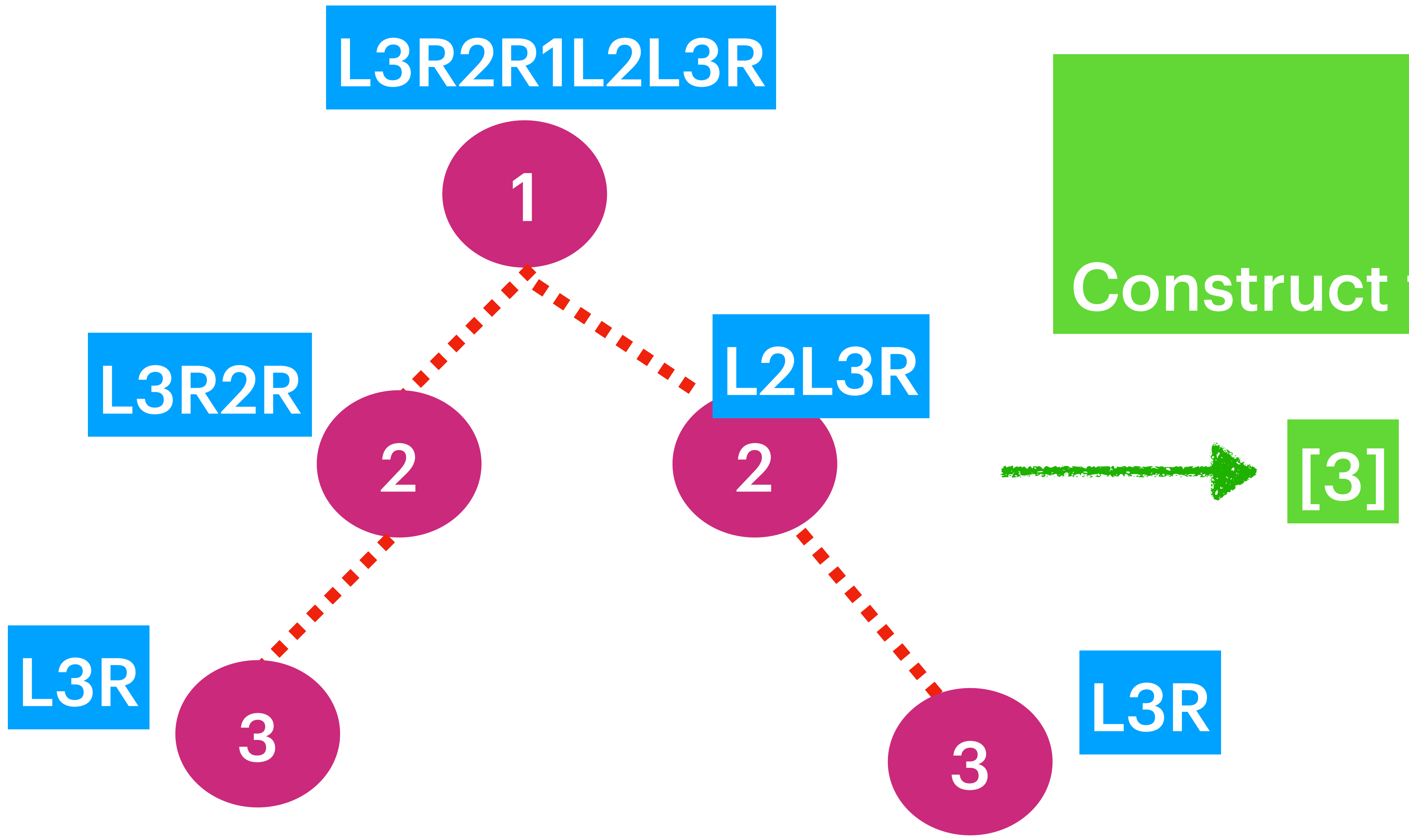


Map<String, TreeNode> key: InOrderString, value: currentNode



Collect the first node in each key where size is > 1

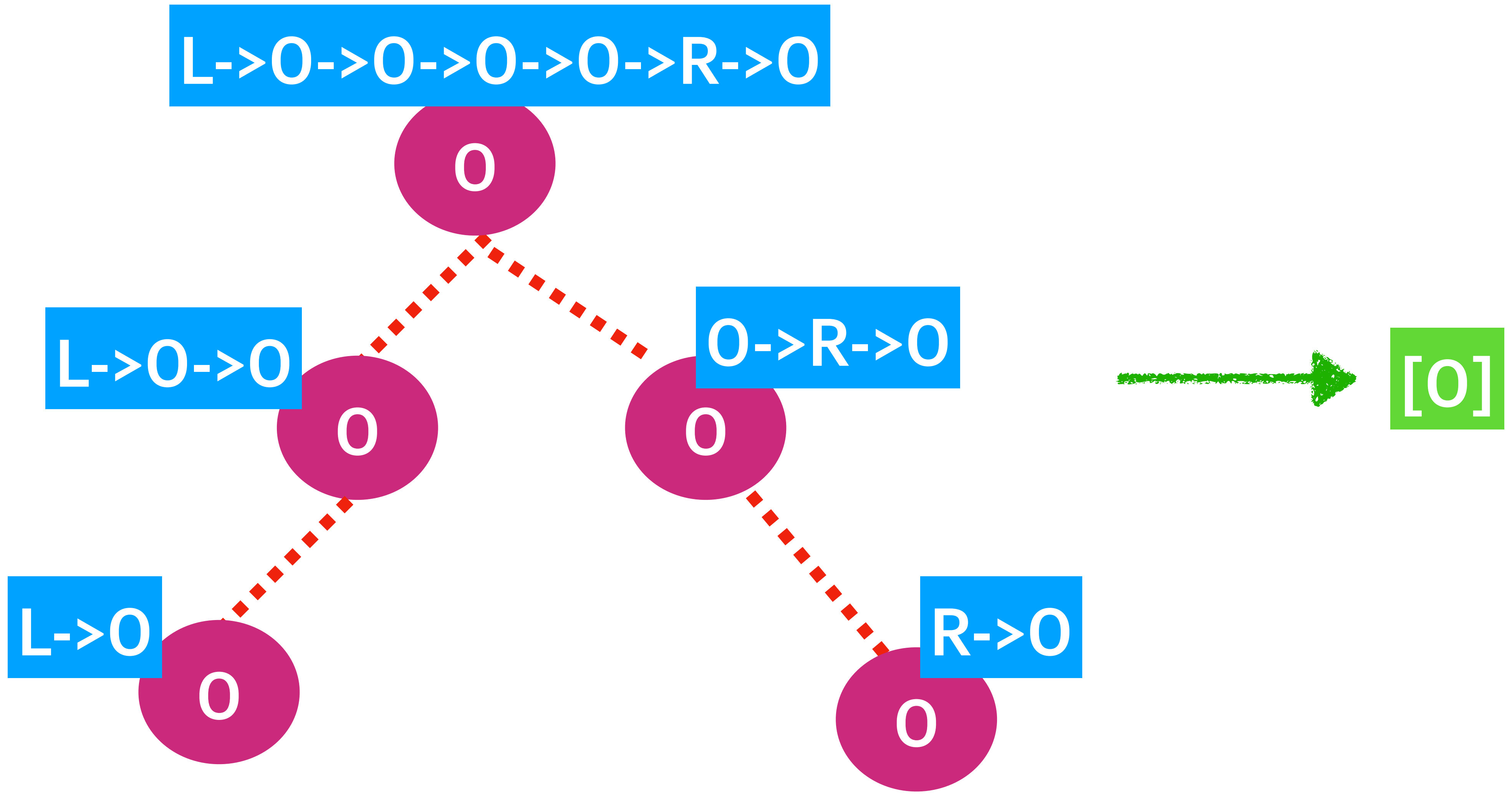




Algorithm :
Do the InOrder Traversal for each Node.
Construct the hashKey based on preOrderString for each node then map it.

Time Complexity: O(n)

Space Complexity: O(n)


L->O->O->O->O->R->O
O
L->O->O
O
O->R->O
O
L->O
O
R->O
O
[O]