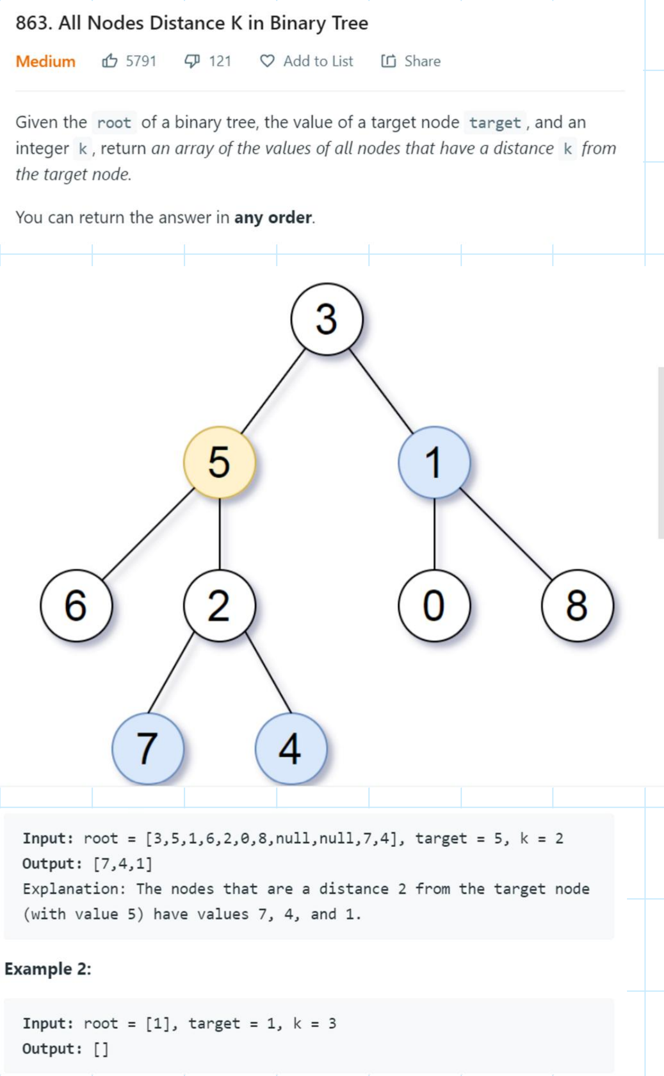
863. All Nodes Distance K in Binary Tree

08 March 2022 10:46 AM



863. All Nodes Distance K in Binary Tree

Medium 5791 121 Add to List Share

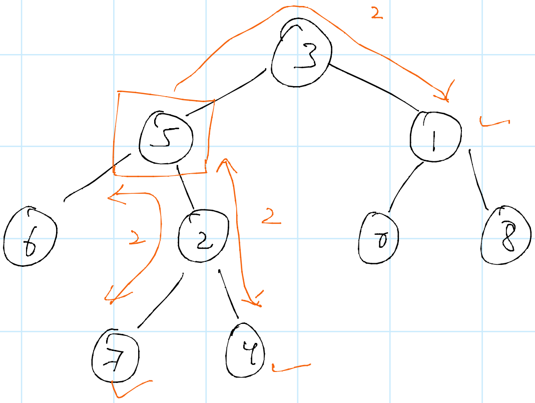
Given the `root` of a binary tree, the value of a target node `target`, and an integer `k`, return *an array of the values of all nodes that have a distance* `k` *from the target node.*

You can return the answer in **any order**.

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], target = 5, k = 2
Output: [7,4,1]
Explanation: The nodes that are a distance 2 from the target node
(with value 5) have values 7, 4, and 1.
```

Example 2:

```
Input: root = [1], target = 1, k = 3
Output: []
```

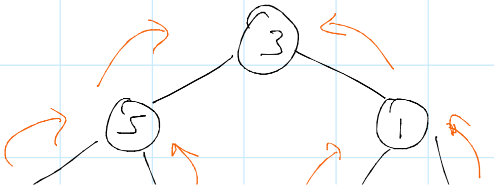


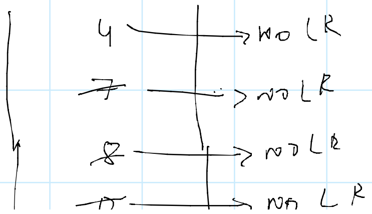
k = 2, target = 5

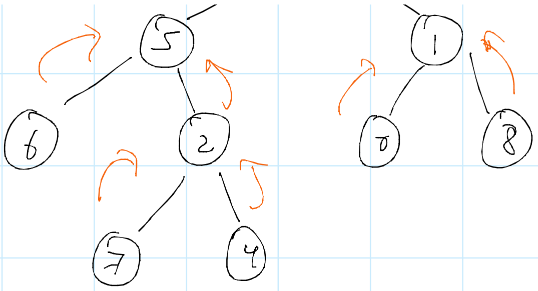
output ⇒ [7, 4, 1]

print in any order

(bfs, traversal) → Queue data struct







8 → no L R
0 → no L R
2 → paint of 7 4
6 → no left and right
1 " " 0 8
5 " " 6 2
3 → parent of 5 1

Q

* first step is to mark the parent node, do that in Hashmap

* Now we can move ↑↓ (to move ↑↓, you need parent node you did that in 1st step)

↑ → upward

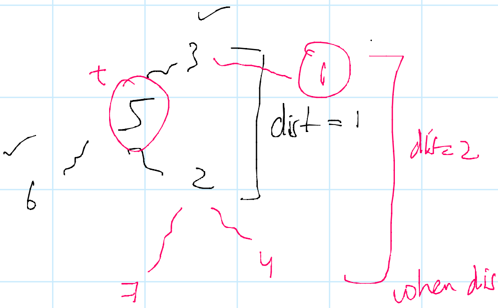
↓ → downward

* BFS traversal from the target node until the k=2 (distance of 2)

and carry a visited hash.

DRY Run

$\boxed{dis = \cancel{0} + 2}$



1
4
7
3
6
2
5

Vis

1
4
7 → ans dis(2)
3
6
2 → any order put and add them in visit
5

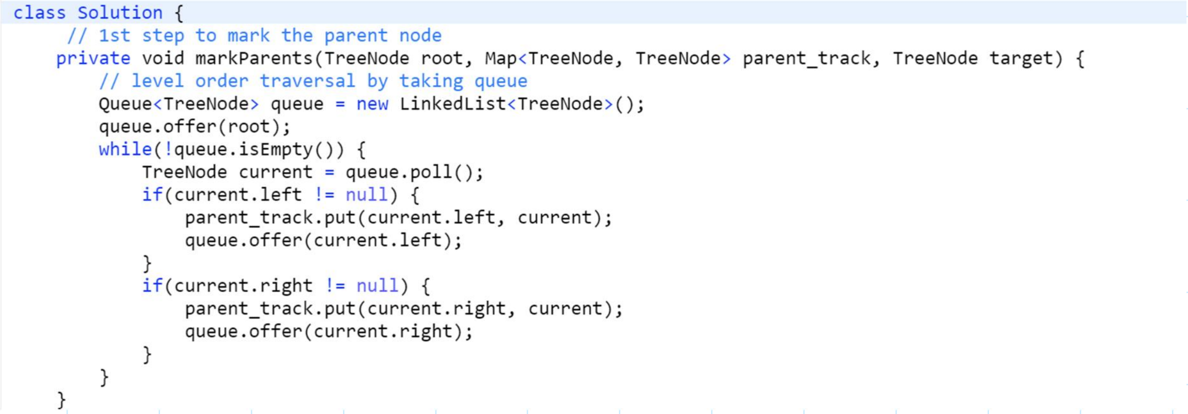
Q

when dist = k=2
So you stop moving upward/downward

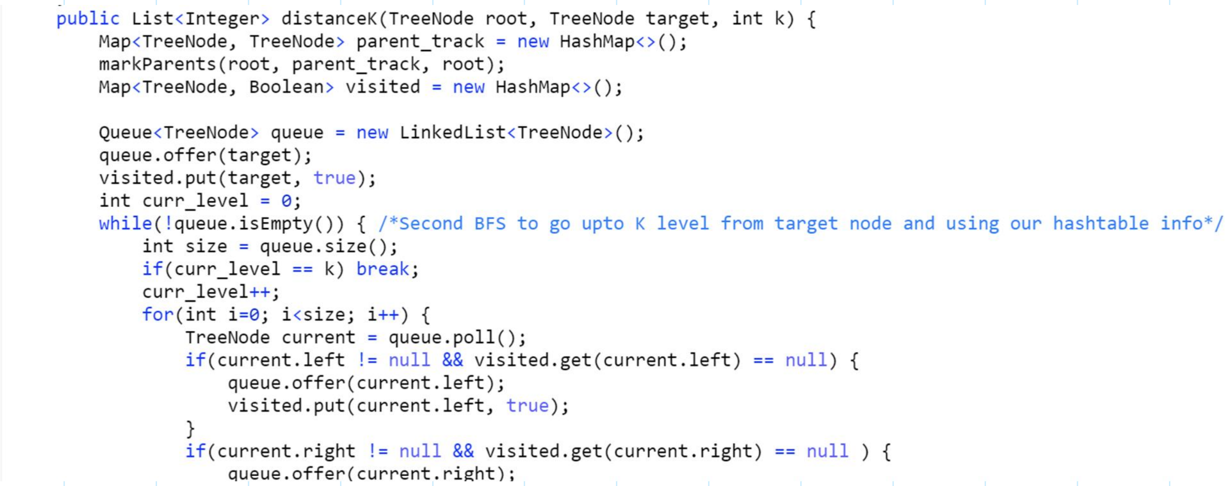
Steps:

- Mark each node to its parent to traverse upwards
- We will do a BFS traversal starting from the target node
- As long as we have not seen our node previously, Traverse up, left, right until reached Kth distance
- when reached Kth distance, break out of BFS loop and remaining node's values in our queue is our result

JAVA Code

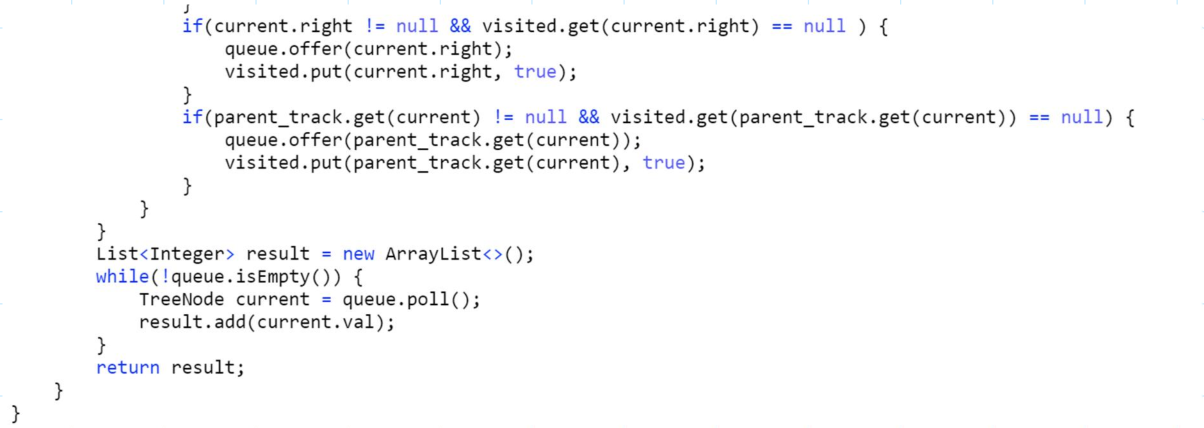


```
class Solution {
     // 1st step to mark the parent node
    private void markParents(TreeNode root, Map<TreeNode, TreeNode> parent_track, TreeNode target) {
        // level order traversal by taking queue
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.offer(root);
        while(!queue.isEmpty()) {
            TreeNode current = queue.poll();
            if(current.left != null) {
                parent_track.put(current.left, current);
                queue.offer(current.left);
            }
            if(current.right != null) {
                parent_track.put(current.right, current);
                queue.offer(current.right);
            }
        }
    }
```



```
    public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {
        Map<TreeNode, TreeNode> parent_track = new HashMap<>();
        markParents(root, parent_track, root);
        Map<TreeNode, Boolean> visited = new HashMap<>();

        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.offer(target);
        visited.put(target, true);
        int curr_level = 0;
        while(!queue.isEmpty()) { /*Second BFS to go upto K level from target node and using our hashtable info*/
            int size = queue.size();
            if(curr_level == k) break;
            curr_level++;
            for(int i=0; i<size; i++) {
                TreeNode current = queue.poll();
                if(current.left != null && visited.get(current.left) == null) {
                    queue.offer(current.left);
                    visited.put(current.left, true);
                }
                if(current.right != null && visited.get(current.right) == null ) {
                    queue.offer(current.right);
```



```
                if(current.right != null && visited.get(current.right) == null ) {
                    queue.offer(current.right);
```

```
                    visited.put(current.right, true);
                }
                if(parent_track.get(current) != null && visited.get(parent_track.get(current)) == null) {
                    queue.offer(parent_track.get(current));
                    visited.put(parent_track.get(current), true);
                }
            }
        }
        List<Integer> result = new ArrayList<>();
        while(!queue.isEmpty()) {
            TreeNode current = queue.poll();
            result.add(current.val);
        }
        return result;
    }
}
```