All Nodes Distance K in BT

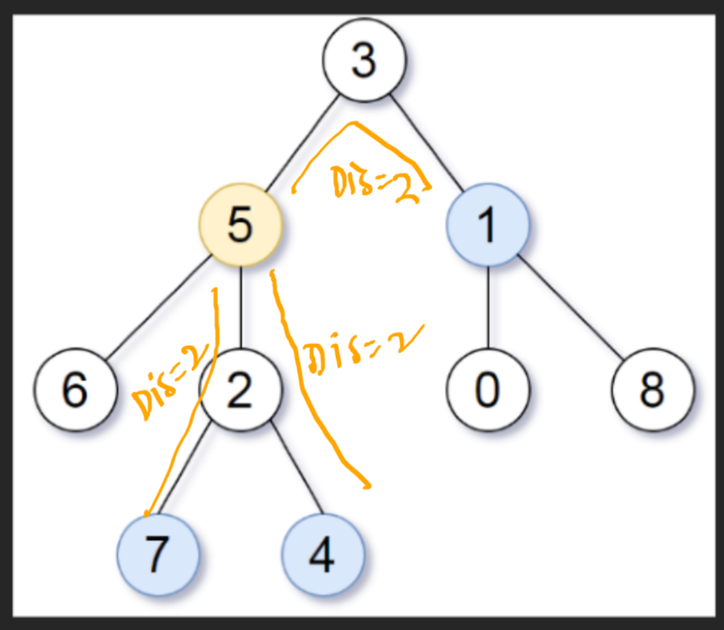
863. All Nodes Distance K in Binary Tree

Medium | Topics | Companies

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**.

Example 1:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], target = 5, k = 2
Output: [7,4,1]
```

* We have to return all the nodes which are K distance from given node

Problem

* We can somehow travel K distance from the give node to down children and return them easily.

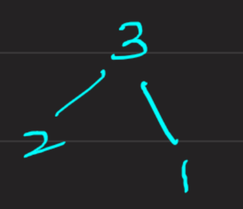
* But the problem is, how to go above

In this question ↑ from 5, how to reach (1)

for that we need to figure out a logic...

* Create a parent map, which store parent info for every node

Ex:-



→ if its a tree

Hashmap

3 → (-1)
2 → 3
1 → 3

→ Hashmap says for node (3), -1 is the parent means null
for node (2), 3 is the parent

* If we have this map, we can easily access the parent of any node

→ InOrder to update every nodes parent in Hashmap, we need to traverse through all nodes,
Do BFS (easy to Update Values)

if you call this function & it will update all parents in hashmap

```java
public void mapParent(TreeNode root,HashMap<TreeNode,TreeNode> parentMap){
    //do simple bfs

    Queue<TreeNode> q = new LinkedList<>();
    q.add(root);
    while(q.size()>0){
        TreeNode rem = q.remove();
        if(rem.left!=null){
            parentMap.put(rem.left,rem);          // (1)
            q.add(rem.left);
        }
        if(rem.right!=null){
            parentMap.put(rem.right,rem);         // (2)
            q.add(rem.right);
        }
    }
}
```

→ New parent HashMap

Simple BFS

* But the key point is if you are at some node (X)

and in BFS, you are about to add children of (X) ....
at that point of time ↓

tell hashMap that → this children's parent is (X)

* I have marked (1) & (2) in the above code, those lines signifies this statement

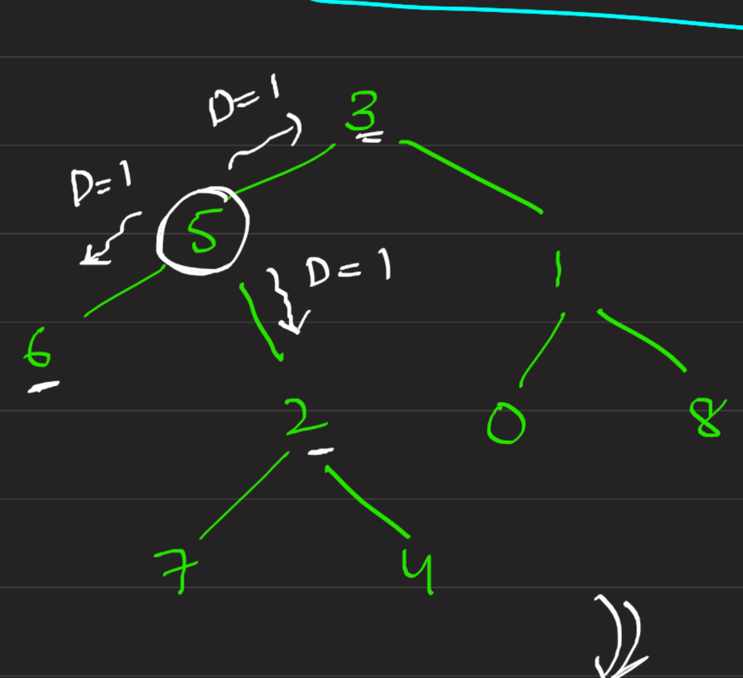
Half of the job is done ↓

so from here on from a given node

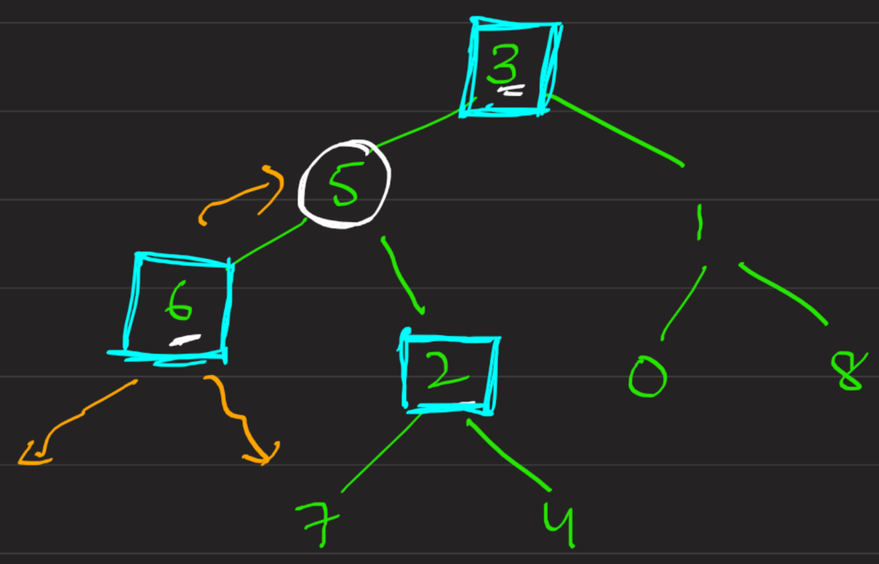
* We can travel to left child
right child
& even parent aswell

* So, from (target) node ↓

travel radially



At one distance,
travel to left,
right,
parent



⇒ from these □ boxed nodes again travel a unit distance,

{ left, right, Parent }

* Here is the edge case :

if you are at 6, you have 3 options for a unit distance

{ left, right, parent }

* But, But, But.... we are coming from (parent) only

So → 6 need to know that, (5) is already visited
i don't need to go there again

* for that, you store visiting data in a hashmap

HashMap → { Node, boolean } → already visited or not

↑ a particular node

```
public List<Integer> distanceK(TreeNode root, TreeNode target, int k) {
    //parent-map
    HashMap<TreeNode,TreeNode> parentMap = new HashMap<>();
    mapParent(root,parentMap);

    //go-left,go-right,go-upward by 1 distance everytime
    HashMap<TreeNode,Boolean> visMap = new HashMap<>();

    Queue<TreeNode> q = new LinkedList<>();
    q.add(target);
    visMap.put(target,true);
    int dis = 0;
```

→ this discussed previously

→ to check whether visited or not.

→ start from target

→ put true to target node, because need not to be visited again

this determines the unit distances

* Again, do BFS (But this time, start from target node)
in queue → add { left, right, parent }

if not visited

"it makes sure's that one every BFS level, dis will increase by 1

and the moment dis equals (k) → we end our BFS

```java
int dis = 0;
while(q.size()>0){
    int n = q.size();
    if(dis == k)break;
    dis++;
    for(int i = 0;i<n;i++){
        TreeNode rem = q.remove();
        if(rem.left!=null && visMap.get(rem.left) == null){
            q.add(rem.left);
            visMap.put(rem.left,true);
        }
        if(rem.right!=null && visMap.get(rem.right) == null){
            q.add(rem.right);
            visMap.put(rem.right,true);
        }
        if(parentMap.get(rem)!=null && visMap.get(parentMap.get(rem)) == null){
            q.add(parentMap.get(rem));
            visMap.put(parentMap.get(rem),true);
        }
    }
}
```

if equals breakOut

On every BFS level increase by 1

if left is available & not visited

add it

make it visited by marking it as true

similar story

Same story for parent aswell

* why 2 loops for levelOrder → I have discussed this in my level Order notes → check it :)

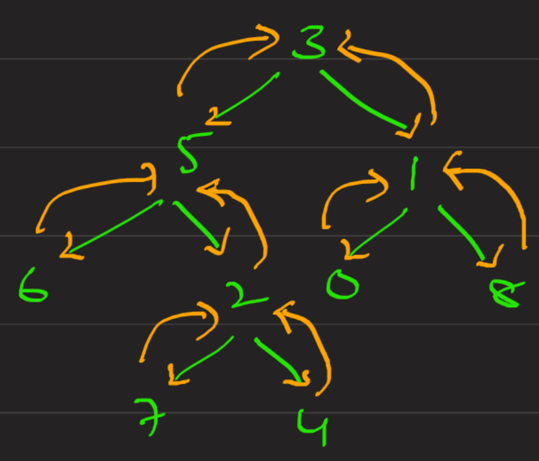
Simply, to specify we are at new level, we use 2 loops

* One loop states → we are at X level
  Inner loop states → we are having n nodes at X - level

→ One each level, add → if possible (left, right, parent)

* Atlast, we have our visited nodes in our queue → at distance = k

Extract from queue & print



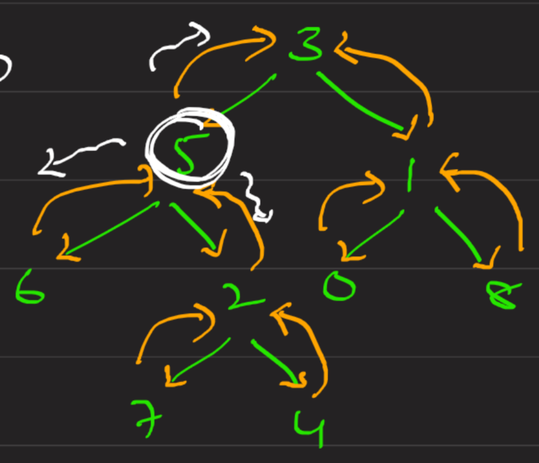
(3, -1)
(5, 3)
(1, 3)
(6, 5)
(2, 5)
(7, 2)
(4, 2)
(0, 1)
(8, 1)

children

parent

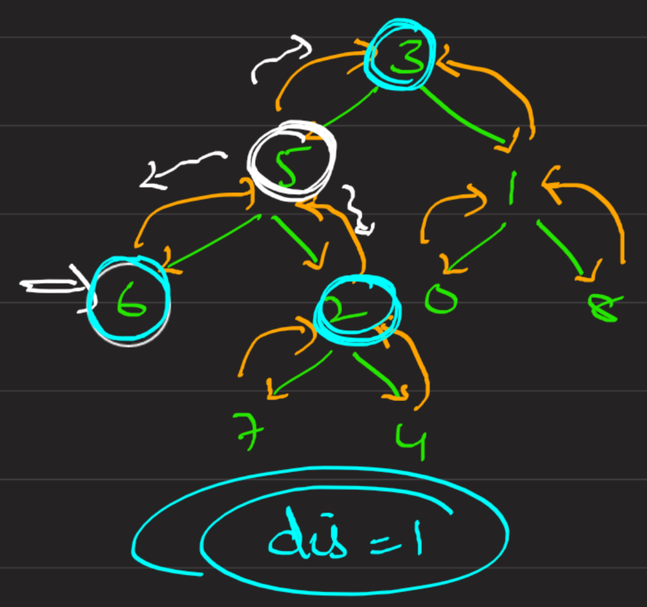
Start at target, go distance = 2

dis=0



(5, true)

Visited Map



~~(5, true)~~

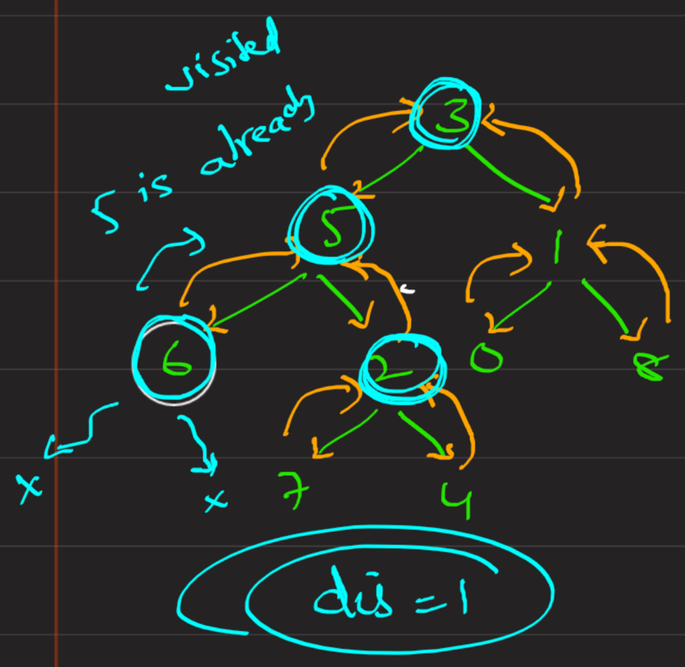
(6, true)

(3, true)

(2, true)

Visited Map

G's → neighbours add



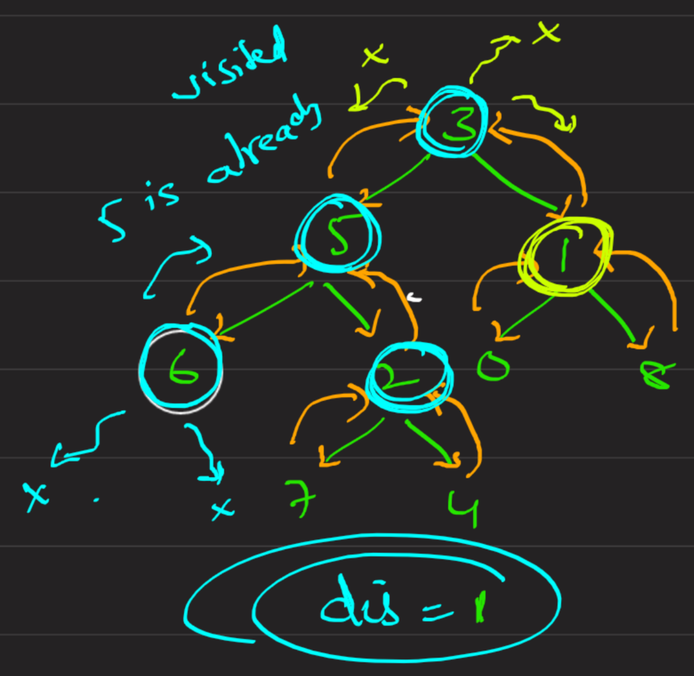
~~(5, true)~~

~~(6, true)~~

(3, true)

(2, true)

3's neighbours



~~(5, true)~~

~~(6, true)~~

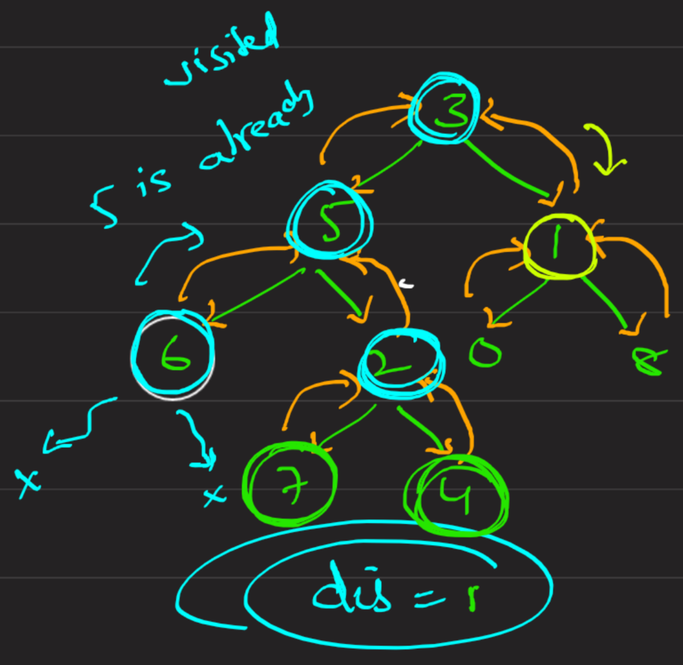
~~(2, true)~~

(2, true)

(1, true)

2's neighbours → after all d=1, neighbours done

d becomes 2, when (d==k) it breaks



D=

~~(5, true)~~ D = 0

~~(6, true)~~

~~(2, true)~~ D = 1

~~(1, true)~~

(1, true)

(7, true)

(4, true)

D = 2

These 3 nodes are at k=2 distance from node

So, we break when we hit dis = 2

So queue will remain with {1, 7, 4} nodes

Add it to your answer