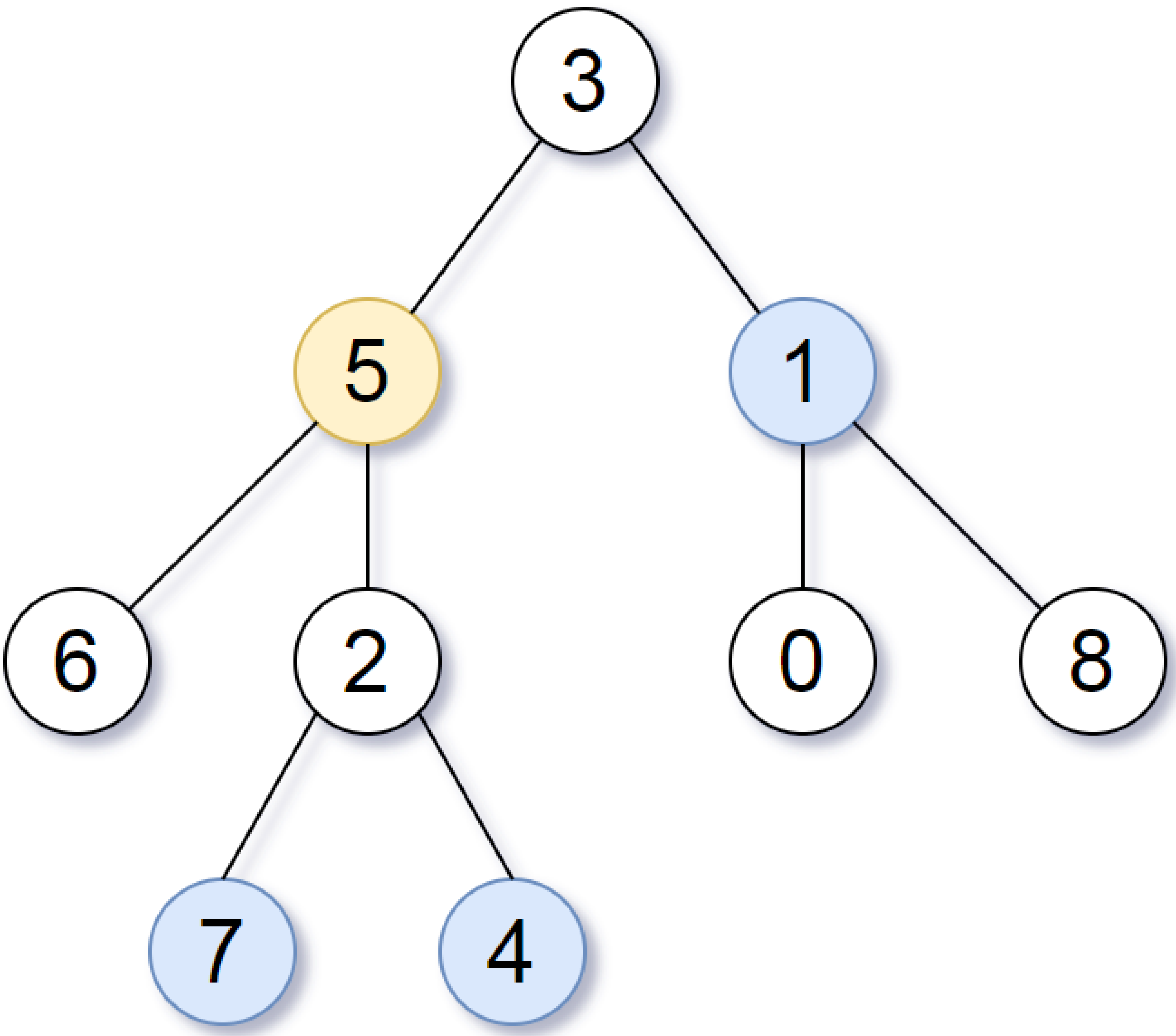
# 863. All Nodes Distance K in Binary Tree

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]
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: []
```

**Constraints:**

- The number of nodes in the tree is in the range `[1, 500]`.
- `0 <= Node.val <= 500`
- All the values `Node.val` are **unique**.
- `target` is the value of one of the nodes in the tree.
- `0 <= k <= 1000`

```
class Solution:
    def distanceK(self, root: TreeNode, target: TreeNode, k: int) -> 
List[int]:
        if root is None:
            return []
        res = []
        self.nodeToRootPath(root,res,target)
        i = 0
        ans = []
        # print(res)
        while i<=k and i<len(res):
            temp = []
            refNode = None if i==0 else res[i-1]
            self.KdownNodes(res[i],refNode,k-i,temp)
            ans = ans+temp
            i+=1
        return ans

    def nodeToRootPath(self,root,res,target):
        if root is None:
            return False
        if root.val == target.val:
            res.append(root)
            return True

        if self.nodeToRootPath(root.left,res,target) or \
        self.nodeToRootPath(root.right,res,target):
            res.append(root)
            return True

        return False
```

```python
    def KdownNodes(self,root,refNode,k,res):
        if root is None or k<0 or root == refNode:
            return
        if k==0:
            res.append(root.val)
            return
        self.KdownNodes(root.left,refNode,k-1,res)
        self.KdownNodes(root.right,refNode,k-1,res)
```

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def distanceK(self, root: TreeNode, target: TreeNode, k: int) ->
List[int]:
        if root is None:
            return []
        ans = []
        self.distnceKHelper(root,target,k,ans)
        return ans


    def distnceKHelper(self,root,target,k,ans):
        if root is None:
            return -1
        if root.val==target.val:
            self.KdownNodes(root,None,k,ans)
            return 1
        ld = self.distnceKHelper(root.left,target,k,ans)
        if ld!=-1:
            self.KdownNodes(root,root.left,k-ld,ans)
            ld=ld+1
            return ld
        rd = self.distnceKHelper(root.right,target,k,ans)
        if rd!=-1:
            self.KdownNodes(root,root.right,k-rd,ans)
            rd=rd+1
            return rd
        return -1




    def KdownNodes(self,root,refNode,k,res):
        if root is None or k<0 or root == refNode:
            return
        if k==0:
            res.append(root.val)
            return
        self.KdownNodes(root.left,refNode,k-1,res)
        self.KdownNodes(root.right,refNode,k-1,res)
```