Given a binary tree **where every node has a unique value**, and a target key k, find the value of the nearest leaf node to target k in the tree.

Here, *nearest* to a leaf means the least number of edges travelled on the binary tree to reach any leaf of the tree. Also, a node is called a *leaf* if it has no children.

In the following examples, the input tree is represented in flattened form row by row. The actual root tree given will be a TreeNode object.

**Example 1:**

**Input:**

root = [1, 3, 2], k = 1

Diagram of binary tree:

1

/ \

3 2

**Output:** 2 (or 3)

**Explanation:** Either 2 or 3 is the nearest leaf node to the target of 1.

**Example 2:**

**Input:**

root = [1], k = 1

**Output:** 1

**Explanation:** The nearest leaf node is the root node itself.

**Example 3:**

**Input:**

root = [1,2,3,4,null,null,null,5,null,6], k = 2

Diagram of binary tree:

1

/ \

2 3

/

4

/

5

/

6

**Output:** 3

**Explanation:** The leaf node with value 3 (and not the leaf node with value 6) is nearest to the node with value 2.

**Note:**

1. root represents a binary tree with at least 1 node and at most 1000 nodes.
2. Every node has a unique node.val in range [1, 1000].
3. There exists some node in the given binary tree for which node.val == k.

Solution :

1. Convert to a graph and do shortest path via BFS.
2. Use DP to store closest leaf in a subtree. Now for every node, calculate the closest node in the entire tree by comparing closest leaf in the subtree vs closest leaf reachable through parent. Root’s closest leaf is a leaf in it’s subtree. This value is passed down to both children of root. Now children of root have subtree closest leaf and closest leaf to root. With this info, they can calculate their closest leaf in the entire tree. Now this information is passed on to their children. So on. Once all this information is collected, traverse the tree to find the ‘k’ th node and look for it’s closest leaf memoized in a map. O(n) run time and O(n) memory.