

## # k<sup>th</sup> Node of a Binary Tree

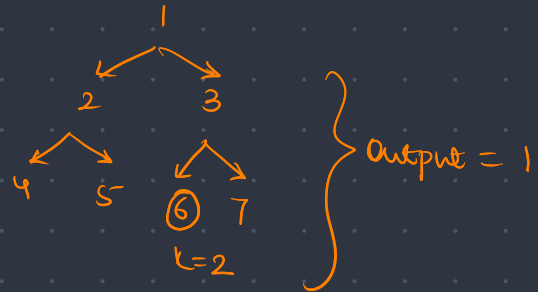
You are given an arbitrary binary tree consisting of N nodes numbered from 1 to N, an integer 'K', and a node 'TARGET\_NODE\_VAL' from the tree. You need to find the Kth ancestor of the node 'TARGET\_NODE\_VAL'. If there is no such ancestor, then print -1.

The Kth ancestor of a node in a binary tree is the Kth node in the path going up from the given node to the root of the tree. Refer to sample test cases for further explanation.

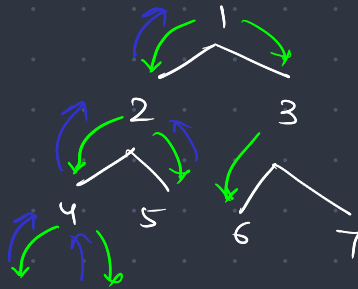
**Note:**

1. The given node 'TARGET\_NODE\_VAL' is always present in the tree.
2. The value of each node in the tree is unique.
3. Notice that the Kth ancestor node if present will always be unique.

→ Brute force:



Approach:



We will push the elements to the array once we traverse through the tree. If we have found the node we stop pushing, going to next nodes or even popping out the elements.

After this to get the required element we simply subtract to get the index.

```

void solve(BinaryTreeNode<int> *root, int targetNodeVal, vector<int> &ans){
    if (root == NULL)
        return;

    ans.push_back(root->data);

    if (root->data == targetNodeVal)
        return;

    solve(root->left, targetNodeVal, ans);
    if (ans.back() != targetNodeVal)
        solve(root->right, targetNodeVal, ans);
    if (ans.back() != targetNodeVal)
        ans.pop_back();
}

int findKthAncestor(BinaryTreeNode<int> *root, int targetNodeVal, int k) {
    vector<int> ans;
    solve(root, targetNodeVal, ans);

    int n = ans.size();

    if (n - 1 - k < 0) return -1;

    return ans[n - 1 - k];
}
    
```