Find the closest leaf in a Binary Tree

Given a Binary Tree and a key 'k', find distance of the closest leaf from 'k'.

Examples:

```
              A
            /    \
           B       C
                 /   \
               E     F
              /       \
             G         H
            / \       /
           I   J     K

Closest leaf to 'H' is 'K', so distance is 1 for 'H'
Closest leaf to 'C' is 'B', so distance is 2 for 'C'
Closest leaf to 'E' is either 'I' or 'J', so distance is 2 for 'E'
Closest leaf to 'B' is 'B' itself, so distance is 0 for 'B'
```

We strongly recommend to minimize your browser and try this yourself first

The main point to note here is that a closest key can either be a descendent of given key or can be reached through one of the ancestors. The idea is to traverse the given tree in preorder and keep track of ancestors in an array. When we reach the given key, we evaluate distance of the closest leaf in subtree rooted with given key. We also traverse all ancestors one by one and find distance of the closest leaf in the subtree rooted with ancestor. We compare all distances and return minimum.

C++

```
// A C++ program to find the closesr leaf of a given key in Binary Tree
#include <iostream>
#include <climits>
using namespace std;

/* A binary tree Node has key, pocharer to left and right children */
struct Node
{
    char key;
    struct Node* left, *right;
};

/* Helper function that allocates a new node with the
   given data and NULL left and right pocharers. */
Node *newNode(char k)
{
    Node *node = new Node;
    node->key = k;
    node->right = node->left = NULL;
    return node;
}

// A utility function to find minimum of x and y
int getMin(int x, int y)
{
    return (x < y)? x :y;
}

// A utility function to find distance of closest leaf of the tree
// rooted under given root
int closestDown(struct Node *root)
{
    // Base cases
    if (root == NULL)
        return INT_MAX;
    if (root->left == NULL && root->right == NULL)
        return 0;

    // Return minimum of left and right, plus one
    return 1 + getMin(closestDown(root->left), closestDown(root->right));
}

// Returns distance of the cloest leaf to a given key 'k'.  The array
// ancestors is used to keep track of ancestors of current node and
// 'index' is used to keep track of curremt index in 'ancestors[]'
int findClosestUtil(struct Node *root, char k, struct Node *ancestors[],
                                               int index)
{
    // Base case
    if (root == NULL)
        return INT_MAX;

    // If key found
    if (root->key == k)
    {
        //  Find the cloest leaf under the subtree rooted with given key
        int res = closestDown(root);

        // Traverse all ancestors and update result if any parent node
        // gives smaller distance
        for (int i = index-1; i>=0; i--)
            res = getMin(res, index - i + closestDown(ancestors[i]));
        return res;
    }

    // If key node found, store current node and recur for left and
    // right childrens
    ancestors[index] = root;
    return getMin(findClosestUtil(root->left, k, ancestors, index+1),
                  findClosestUtil(root->right, k, ancestors, index+1));

}

// The main function that returns distance of the closest key to 'k'. It
// mainly uses recursive function findClosestUtil() to find the closes
// distance.
int findClosest(struct Node *root, char k)
{
    // Create an array to store ancestors
    // Assumptiom: Maximum height of tree is 100
    struct Node *ancestors[100];

    return findClosestUtil(root, k, ancestors, 0);
}

/* Driver program to test above functions*/
int main()
{
    // Let us construct the BST shown in the above figure
    struct Node *root        = newNode('A');
    root->left               = newNode('B');
    root->right              = newNode('C');
    root->right->left        = newNode('E');
    root->right->right       = newNode('F');
    root->right->left->left  = newNode('G');
    root->right->left->left->left  = newNode('I');
    root->right->left->left->right = newNode('J');
    root->right->right->right      = newNode('H');
    root->right->right->right->left = newNode('K');

    char k = 'H';
    cout << "Distace of the closest key from " << k << " is "
         << findClosest(root, k) << endl;
    k = 'C';
    cout << "Distace of the closest key from " << k << " is "
         << findClosest(root, k) << endl;
    k = 'E';
    cout << "Distace of the closest key from " << k << " is "
         << findClosest(root, k) << endl;
    k = 'B';
    cout << "Distace of the closest key from " << k << " is "
         << findClosest(root, k) << endl;

    return 0;
}
```

Java

```
// Java program to find closest leaf of a given key in Binary Tree

/* Class containing left and right child of current
   node and key value*/
class Node
{
    int data;
    Node left, right;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree
{
    Node root;

    // A utility function to find minimum of x and y
    int getMin(int x, int y)
    {
        return (x < y) ? x : y;
    }

    // A utility function to find distance of closest leaf of the tree
    // rooted under given root
    int closestDown(Node node)
    {
        // Base cases
        if (node == null)
            return Integer.MAX_VALUE;
        if (node.left == null && node.right == null)
            return 0;

        // Return minimum of left and right, plus one
        return 1 + getMin(closestDown(node.left), closestDown(node.right));
    }

    // Returns distance of the cloest leaf to a given key 'k'.  The array
    // ancestors is used to keep track of ancestors of current node and
    // 'index' is used to keep track of curremt index in 'ancestors[]'
    int findClosestUtil(Node node, char k, Node ancestors[], int index)
    {
        // Base case
        if (node == null)
            return Integer.MAX_VALUE;

        // If key found
        if (node.data == k)
        {
            //  Find the cloest leaf under the subtree rooted with given key
            int res = closestDown(node);

            // Traverse all ancestors and update result if any parent node
            // gives smaller distance
            for (int i = index - 1; i >= 0; i--)
```

```
                res = getMin(res, index - i + closestDown(ancestors[i]));
            return res;
        }

        // If key node found, store current node and recur for left and
        // right childrens
        ancestors[index] = node;
        return getMin(findClosestUtil(node.left, k, ancestors, index + 1),
                findClosestUtil(node.right, k, ancestors, index + 1));

    }

    // The main function that returns distance of the closest key to 'k'. It
    // mainly uses recursive function findClosestUtil() to find the closes
    // distance.
    int findClosest(Node node, char k)
    {
        // Create an array to store ancestors
        // Assumptiom: Maximum height of tree is 100
        Node ancestors[] = new Node[100];

        return findClosestUtil(node, k, ancestors, 0);
    }

    // Driver program to test for above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node('A');
        tree.root.left = new Node('B');
        tree.root.right = new Node('C');
        tree.root.right.left = new Node('E');
        tree.root.right.right = new Node('F');
        tree.root.right.left.left = new Node('G');
        tree.root.right.left.left.left = new Node('I');
        tree.root.right.left.left.right = new Node('J');
        tree.root.right.right.right = new Node('H');
        tree.root.right.right.right.left = new Node('H');

        char k = 'H';
        System.out.println("Distace of the closest key from " + k + " is "
                            + tree.findClosest(tree.root, k));
        k = 'C';
        System.out.println("Distace of the closest key from " + k + " is "
                            + tree.findClosest(tree.root, k));
        k = 'E';
        System.out.println("Distace of the closest key from " + k + " is "
                            + tree.findClosest(tree.root, k));
        k = 'B';
        System.out.println("Distace of the closest key from " + k + " is "
                             + tree.findClosest(tree.root, k));

    }
}

// This code has been contributed by Mayank Jaiswal
```

Python

```
# Python program to find closest leaf of a
# given key in binary tree

INT_MAX = 2**32

# A binary tree node
class Node:
    # Constructor to create a binary tree
    def __init__(self ,key):
        self.key = key
        self.left  = None
        self.right = None

def closestDown(root):
    #Base Case
    if root is None:
        return INT_MAX
    if root.left is None and root.right is None:
        return 0

    # Return minum of left and right plus one
    return 1 + min(closestDown(root.left),
                   closestDown(root.right))

# Returns destance of the closes leaf to a given key k
# The array ancestors us used to keep track of ancestors
# of current node and 'index' is used to keep track of
# current index in 'ancestors[i]'
def findClosestUtil(root, k, ancestors, index):
    # Base Case
    if root is None:
        return INT_MAX

    # if key found
    if root.key == k:
        # Find closest leaf under the subtree rooted
        # with given key
        res = closestDown(root)

        # Traverse ll ancestors and update result if any
        # parent node gives smaller distance
        for i in reversed(range(0,index)):
            res = min(res, index-i+closestDown(ancestors[i]))
        return res

    # if key node found, store current node and recur for left
    # and right childrens
    ancestors[index] = root
    return min(
        findClosestUtil(root.left, k,ancestors, index+1),
        findClosestUtil(root.right, k, ancestors, index+1))

# The main function that return distance of the clses key to
# 'key'. It mainly uses recursive function findClosestUtil()
# to find the closes distance
def findClosest(root, k):
    # Create an arrray to store ancestors
    # Assumption: Maximum height of tree is 100
    ancestors = [None for i in range(100)]

    return findClosestUtil(root, k, ancestors, 0)


# Driver program to test above function
root = Node('A')
root.left = Node('B')
root.right = Node('C');
root.right.left = Node('E');
root.right.right  = Node('F');
root.right.left.left = Node('G');
root.right.left.left.left  = Node('I');
root.right.left.left.right = Node('J');
root.right.right.right  = Node('H');
root.right.right.right.left = Node('K');

k = 'H';
print "Distance of the closest key from "+ k + " is",
print findClosest(root, k)

k = 'C'
print "Distance of the closest key from " + k + " is",
print findClosest(root, k)
```

```
k = 'E'
print "Distance of the closest key from " + k + " is",
print findClosest(root, k)

k = 'B'
print "Distance of the closest key from " + k + " is",
print findClosest(root, k)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
Distace of the closest key from H is 1
Distace of the closest key from C is 2
Distace of the closest key from E is 2
Distace of the closest key from B is 0
```

The above code can be optimized by storing the left/right information also in ancestor array. The idea is, if given key is in left subtree of an ancestors, then there is no point to call closestDown(). Also, the loop can that traverses ancestors array can be optimized to not traverse ancestors which are at more distance than current result.

Exercise:
Extend the above solution to print not only distance, but the key of closest leaf also.