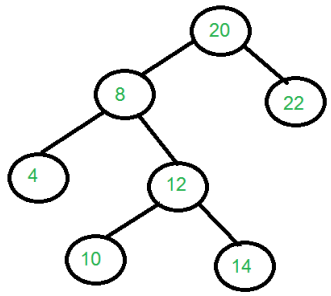
Print all nodes at distance k from a given node

Given a binary tree, a target node in the binary tree, and an integer value k, print all the nodes that are at distance k from the given target node. No parent pointers are available.




```
Consider the tree shown in diagram

Input: target = pointer to node with data 8.
       root = pointer to node with data 20.
       k = 2.
Output : 10 14 22

If target is 14 and k is 3, then output
should be "4 20"
```

We strongly recommend that you click here and practice it, before moving on to the solution.

There are two types of nodes to be considered.
1) Nodes in the subtree rooted with target node. For example if the target node is 8 and k is 2, then such nodes are 10 and 14.
2) Other nodes, may be an ancestor of target, or a node in some other subtree. For target node 8 and k is 2, the node 22 comes in this category.

Finding the first type of nodes is easy to implement. Just traverse subtrees rooted with the target node and decrement k in recursive call. When the k becomes 0, print the node currently being traversed (See this for more details). Here we call the function as *printkdistanceNodeDown()*.

How to find nodes of second type? For the output nodes not lying in the subtree with the target node as the root, we must go through all ancestors. For every ancestor, we find its distance from target node, let the distance be d, now we go to other subtree (if target was found in left subtree, then we go to right subtree and vice versa) of the ancestor and find all nodes at k-d distance from the ancestor.

Following is the implementation of the above approach.

C++

```
#include <iostream>
using namespace std;

// A binary Tree node
struct node
{
    int data;
    struct node *left, *right;
};

/* Recursive function to print all the nodes at distance k in the
   tree (or subtree) rooted with given root. See  */
void printkdistanceNodeDown(node *root, int k)
{
    // Base Case
    if (root == NULL || k < 0)  return;

    // If we reach a k distant node, print it
    if (k==0)
    {
        cout << root->data << endl;
        return;
    }
```

```
    }

    // Recur for left and right subtrees
    printkdistanceNodeDown(root->left, k-1);
    printkdistanceNodeDown(root->right, k-1);
}

// Prints all nodes at distance k from a given target node.
// The k distant nodes may be upward or downward.  This function
// Returns distance of root from target node, it returns -1 if target
// node is not present in tree rooted with root.
int printkdistanceNode(node* root, node* target , int k)
{
    // Base Case 1: If tree is empty, return -1
    if (root == NULL) return -1;

    // If target is same as root.  Use the downward function
    // to print all nodes at distance k in subtree rooted with
    // target or root
    if (root == target)
    {
        printkdistanceNodeDown(root, k);
        return 0;
    }

    // Recur for left subtree
    int dl = printkdistanceNode(root->left, target, k);

    // Check if target node was found in left subtree
    if (dl != -1)
    {
         // If root is at distance k from target, print root
         // Note that dl is Distance of root's left child from target
         if (dl + 1 == k)
            cout << root->data << endl;

         // Else go to right subtree and print all k-dl-2 distant nodes
         // Note that the right child is 2 edges away from left child
         else
            printkdistanceNodeDown(root->right, k-dl-2);

         // Add 1 to the distance and return value for parent calls
         return 1 + dl;
    }

    // MIRROR OF ABOVE CODE FOR RIGHT SUBTREE
    // Note that we reach here only when node was not found in left subtree
    int dr = printkdistanceNode(root->right, target, k);
    if (dr != -1)
    {
         if (dr + 1 == k)
            cout << root->data << endl;
         else
            printkdistanceNodeDown(root->left, k-dr-2);
         return 1 + dr;
    }

    // If target was neither present in left nor in right subtree
    return -1;
}

// A utility function to create a new binary tree node
node *newnode(int data)
{
    node *temp = new node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Driver program to test above functions
int main()
{
```

```
    /* Let us construct the tree shown in above diagram */
    node * root = newnode(20);
    root->left = newnode(8);
    root->right = newnode(22);
    root->left->left = newnode(4);
    root->left->right = newnode(12);
    root->left->right->left = newnode(10);
    root->left->right->right = newnode(14);
    node * target = root->left->right;
    printkdistanceNode(root, target, 2);
    return 0;
}
```

Java

```
// Java program to print all nodes at a distance k from given node

// A binary tree node
class Node
{
    int data;
    Node left, right;

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

class BinaryTree
{
    Node root;
    /* Recursive function to print all the nodes at distance k in
       tree (or subtree) rooted with given root. */

    void printkdistanceNodeDown(Node node, int k)
    {
        // Base Case
        if (node == null || k < 0)
            return;

        // If we reach a k distant node, print it
        if (k == 0)
        {
            System.out.print(node.data);
            System.out.println("");
            return;
        }

        // Recur for left and right subtrees
        printkdistanceNodeDown(node.left, k - 1);
        printkdistanceNodeDown(node.right, k - 1);
    }

    // Prints all nodes at distance k from a given target node.
    // The k distant nodes may be upward or downward.This function
    // Returns distance of root from target node, it returns -1
    // if target node is not present in tree rooted with root.
    int printkdistanceNode(Node node, Node target, int k)
    {
        // Base Case 1: If tree is empty, return -1
        if (node == null)
            return -1;

        // If target is same as root.  Use the downward function
        // to print all nodes at distance k in subtree rooted with
        // target or root
        if (node == target)
        {
```

```
            printkdistanceNodeDown(node, k);
            return 0;
        }

        // Recur for left subtree
        int dl = printkdistanceNode(node.left, target, k);

        // Check if target node was found in left subtree
        if (dl != -1)
        {

            // If root is at distance k from target, print root
            // Note that dl is Distance of root's left child from
            // target
            if (dl + 1 == k)
            {
                System.out.print(node.data);
                System.out.println("");
            }

            // Else go to right subtree and print all k-dl-2 distant nodes
            // Note that the right child is 2 edges away from left child
            else
                printkdistanceNodeDown(node.right, k - dl - 2);

            // Add 1 to the distance and return value for parent calls
            return 1 + dl;
        }

        // MIRROR OF ABOVE CODE FOR RIGHT SUBTREE
        // Note that we reach here only when node was not found in left
        // subtree
        int dr = printkdistanceNode(node.right, target, k);
        if (dr != -1)
        {
            if (dr + 1 == k)
            {
                System.out.print(node.data);
                System.out.println("");
            }
            else
                printkdistanceNodeDown(node.left, k - dr - 2);
            return 1 + dr;
        }

        // If target was neither present in left nor in right subtree
        return -1;
    }

    // Driver program to test the above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();

        /* Let us construct the tree shown in above diagram */
        tree.root = new Node(20);
        tree.root.left = new Node(8);
        tree.root.right = new Node(22);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(12);
        tree.root.left.right.left = new Node(10);
        tree.root.left.right.right = new Node(14);
        Node target = tree.root.left.right;
        tree.printkdistanceNode(tree.root, target, 2);
    }
}

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

Python

```
# Python program to print nodes at distance k from a given node

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

# Recursive function to print all the nodes at distance k
# int the tree(or subtree) rooted with given root. See
def printkDistanceNodeDown(root, k):

    # Base Case
    if root is None or k< 0 :
        return

    # If we reach a k distant node, print it
    if k == 0 :
        print root.data
        return

    # Recur for left and right subtee
    printkDistanceNodeDown(root.left, k-1)
    printkDistanceNodeDown(root.right, k-1)


# Prints all nodes at distance k from a given target node
# The k distant nodes may be upward or downward. This function
# returns distance of root from target node, it returns -1
# if target node is not present in tree rooted with root
def printkDistanceNode(root, target, k):

    # Base Case 1 : IF tree is empty return -1
    if root is None:
        return -1

    # If target is same as root. Use the downward function
    # to print all nodes at distance k in subtree rooted with
    # target or root
    if root == target:
        printkDistanceNodeDown(root, k)
        return 0

    # Recur for left subtree
    dl = printkDistanceNode(root.left, target, k)

    # Check if target node was found in left subtree
    if dl != -1:

        # If root is at distance k from target, print root
        # Note: dl is distance of root's left child
        # from target
        if dl +1 == k :
            print root.data

        # Else go to right subtreee and print all k-dl-2
        # distant nodes
        # Note: that the right child is 2 edges away from
        # left chlid
        else:
            printkDistanceNodeDown(root.right, k-dl-2)

        # Add 1 to the distance and return value for
        # for parent calls
        return 1 + dl

    # MIRROR OF ABOVE CODE FOR RIGHT SUBTREE
    # Note that we reach here only when node was not found
    # in left subtree
```

```
    dr = printkDistanceNode(root.right, target, k)
    if dr != -1:
        if (dr+1 == k):
            print root.data
        else:
            printkDistanceNodeDown(root.left, k-dr-2)
        return 1 + dr

    # If target was neither present in left nor in right subtree
    return -1

# Driver program to test above function
root = Node(20)
root.left = Node(8)
root.right = Node(22)
root.left.left = Node(4)
root.left.right = Node(12)
root.left.right.left = Node(10)
root.left.right.right = Node(14)
target = root.left.right
printkDistanceNode(root, target, 2)

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

Output:

```
4
20
```

Time Complexity: At first look the time complexity looks more than O(n), but if we take a closer look, we can observe that no node is traversed more than twice. Therefore the time complexity is O(n).