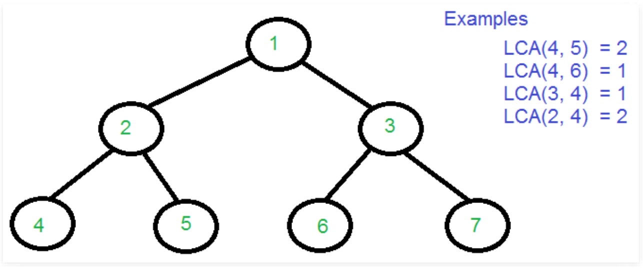
# Lowest Common Ancestor in a Binary Tree | Set 1

Given a binary tree (not a binary search tree) and two values say n1 and n2, write a program to find the least common ancestor.

***Following is definition of LCA from Wikipedia:***

Let T be a rooted tree. The lowest common ancestor between two nodes n1 and n2 is defined as the lowest node in T that has both n1 and n2 as descendants (where we allow a node to be a descendant of itself).

The LCA of n1 and n2 in T is the shared ancestor of n1 and n2 that is located farthest from the root. Computation of lowest common ancestors may be useful, for instance, as part of a procedure for determining the distance between pairs of nodes in a tree: the distance from n1 to n2 can be computed as the distance from the root to n1, plus the distance from the root to n2, minus twice the distance from the root to their lowest common ancestor. (Source Wiki)



**Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.**

We have discussed an efficient solution to find LCA in Binary Search Tree. In Binary Search Tree, using BST properties, we can find LCA in O(h) time where h is height of tree. Such an implementation is not possible in Binary Tree as keys Binary Tree nodes don't follow any order. Following are different approaches to find LCA in Binary Tree.

**Method 1 (By Storing root to n1 and root to n2 paths):**
Following is simple O(n) algorithm to find LCA of n1 and n2.
**1)** Find path from root to n1 and store it in a vector or array.
**2)** Find path from root to n2 and store it in another vector or array.
**3)** Traverse both paths till the values in arrays are same. Return the common element just before the mismatch.

Following is C++ implementation of above algorithm.

C++

Java

Python

```
# O(n) solution to find LCS of two given values n1 and n2

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

# Finds the path from root node to given root of the tree.
# Stores the path in a list path[], returns true if path
# exists otherwise false
def findPath( root, path, k):

    # Baes Case
    if root is None:
        return False

    # Store this node is path vector. The node will be
    # removed if not in path from root to k
    path.append(root.key)

    # See if the k is same as root's key
    if root.key == k :
        return True

    # Check if k is found in left or right sub-tree
    if ((root.left != None and findPath(root.left, path, k)) or
            (root.right!= None and findPath(root.right, path, k))):
        return True
    # If not present in subtree rooted with root, remove
    # root from path and return False

    path.pop()
    return False

# Returns LCA if node n1 , n2 are present in the given
# binary tre otherwise return -1
def findLCA(root, n1, n2):

    # To store paths to n1 and n2 fromthe root
    path1 = []
    path2 = []

    # Find paths from root to n1 and root to n2.
    # If either n1 or n2 is not present , return -1
    if (not findPath(root, path1, n1) or not findPath(root, path2, n2)):
        return -1

    # Compare the paths to get the first different value
    i = 0
    while(i < len(path1) and i < len(path2)):
        if path1[i] != path2[i]:
            break
        i += 1
    return path1[i-1]


# Driver program to test above function
# Let's create the Binary Tree shown in above diagram
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

print "LCA(4, 5) = %d" %(findLCA(root, 4, 5,))
print "LCA(4, 6) = %d" %(findLCA(root, 4, 6))
print "LCA(3, 4) = %d" %(findLCA(root,3,4))
print "LCA(2, 4) = %d" %(findLCA(root,2, 4))

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

Run on IDE

Output:

```
LCA(4, 5) = 2
LCA(4, 6) = 1
LCA(3, 4) = 1
LCA(2, 4) = 2
```

***Time Complexity:*** Time complexity of the above solution is O(n). The tree is traversed twice, and then path arrays are compared.
Thanks to *Ravi Chandra Enaganti* for suggesting the initial solution based on this method.

**Method 2 (Using Single Traversal)**
The method 1 finds LCA in O(n) time, but requires three tree traversals plus extra spaces for path arrays. If we assume that the keys n1 and n2 are present in Binary Tree, we can find LCA using single traversal of Binary Tree and without extra storage for path arrays.
The idea is to traverse the tree starting from root. If any of the given keys (n1 and n2) matches with root, then root is LCA (assuming that both keys are present). If root doesn't match with any of the keys, we recur for left and right subtree. The node which has one key present in its left subtree and the other key present in right subtree is the LCA. If both keys lie in left subtree, then left subtree has LCA also, otherwise LCA lies in right subtree.

C++

Java

Python

```
# Python program to find LCA of n1 and n2 using one
# traversal of Binary tree

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

# This function returns pointer to LCA of two given
# values n1 and n2
# This function assumes that n1 and n2 are present in
# Binary Tree
def findLCA(root, n1, n2):

    # Base Case
    if root is None:
        return None

    # If either n1 or n2 matches with root's key, report
    #  the presence by returning root (Note that if a key is
    #  ancestor of other, then the ancestor key becomes LCA
    if root.key == n1 or root.key == n2:
        return root

    # Look for keys in left and right subtrees
    left_lca = findLCA(root.left, n1, n2)
    right_lca = findLCA(root.right, n1, n2)

    # If both of the above calls return Non-NULL, then one key
    # is present in once subtree and other is present in other,
    # So this node is the LCA
    if left_lca and right_lca:
        return root

    # Otherwise check if left subtree or right subtree is LCA
    return left_lca if left_lca is not None else right_lca


# Driver program to test above function

# Let us create a binary tree given in the above example
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)
print "LCA(4,5) = ", findLCA(root, 4, 5).key
print "LCA(4,6) = ", findLCA(root, 4, 6).key
print "LCA(3,4) = ", findLCA(root, 3, 4).key
print "LCA(2,4) = ", findLCA(root, 2, 4).key

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

Run on IDE

Output:

```
LCA(4, 5) = 2
LCA(4, 6) = 1
LCA(3, 4) = 1
LCA(2, 4) = 2
```

Thanks to *Atul Singh* for suggesting this solution.

***Time Complexity:*** Time complexity of the above solution is O(n) as the method does a simple tree traversal in bottom up fashion.
Note that the above method assumes that keys are present in Binary Tree. If one key is present and other is absent, then it returns the present key as LCA (Ideally should have returned NULL).
We can extend this method to handle all cases by passing two boolean variables v1 and v2. v1 is set as true when n1 is present in tree and v2 is set as true if n2 is present in tree.

C++

Java

Python

```
""" Program to find LCA of n1 and n2 using one traversal of
 Binary tree
It handles all cases even when n1 or n2 is not there in tree
"""

# A binary tree node
class Node:

    # Constructor to create a new node
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

# This function retturn pointer to LCA of two given values
# n1 and n2
# v1 is set as true by this function if n1 is found
# v2 is set as true by this function if n2 is found
def findLCAUtil(root, n1, n2, v):

    # Base Case
    if root is None:
        return None

    # IF either n1 or n2 matches ith root's key, report
    # the presence by setting v1 or v2 as true and return
    # root (Note that if a key is ancestor of other, then
    # the ancestor key becomes LCA)
    if root.key == n1 :
        v[0] = True
        return root

    if root.key == n2:
        v[1] = True
        return root

    # Look for keys in left and right subtree
    left_lca = findLCAUtil(root.left, n1, n2, v)
    right_lca = findLCAUtil(root.right, n1, n2, v)

    # If both of the above calls return Non-NULL, then one key
    # is present in once subtree and other is present in other,
    # So this node is the LCA
    if left_lca and right_lca:
        return root

    # Otherwise check if left subtree or right subtree is LCA
    return left_lca if left_lca is not None else right_lca


def find(root, k):

    # Base Case
    if root is None:
        return False

    # If key is present at root, or if left subtree or right
    # subtree , return true
    if (root.key == k or find(root.left, k) or
        find(root.right, k)):
        return True

    # Else return false
    return False

# This function returns LCA of n1 and n2 onlue if both
# n1 and n2 are present in tree, otherwise returns None
def findLCA(root, n1, n2):

    # Initialize n1 and n2 as not visited
    v = [False, False]

    # Find lac of n1 and n2 using the technique discussed above
    lca = findLCAUtil(root, n1, n2, v)

    # Returns LCA only if both n1 and n2 are present in tree
    if (v[0] and v[1] or v[0] and find(lca, n2) or v[1] and
        find(lca, n1)):
        return lca

    # Else return None
    return None

# Driver program to test above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)

lca = findLCA(root, 4, 5)

if lca is not None:
    print "LCA(4, 5) = ", lca.key
else :
    print "Keys are not present"

lca = findLCA(root, 4, 10)
if lca is not None:
    print "LCA(4,10) = ", lca.key
else:
    print "Keys are not present"

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

Run on IDE

Output:

```
LCA(4, 5) = 2
Keys are not present
```

Thanks to Dhruv for suggesting this extended solution.

**Asked in: Amazon, Expedia, Microsoft, Payu, Snapdeal, Times Internet, Twitter**

You may like to see below articles as well :

LCA using Parent Pointer

Lowest Common Ancestor in a Binary Search Tree.

Find LCA in Binary Tree using RMQ

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above