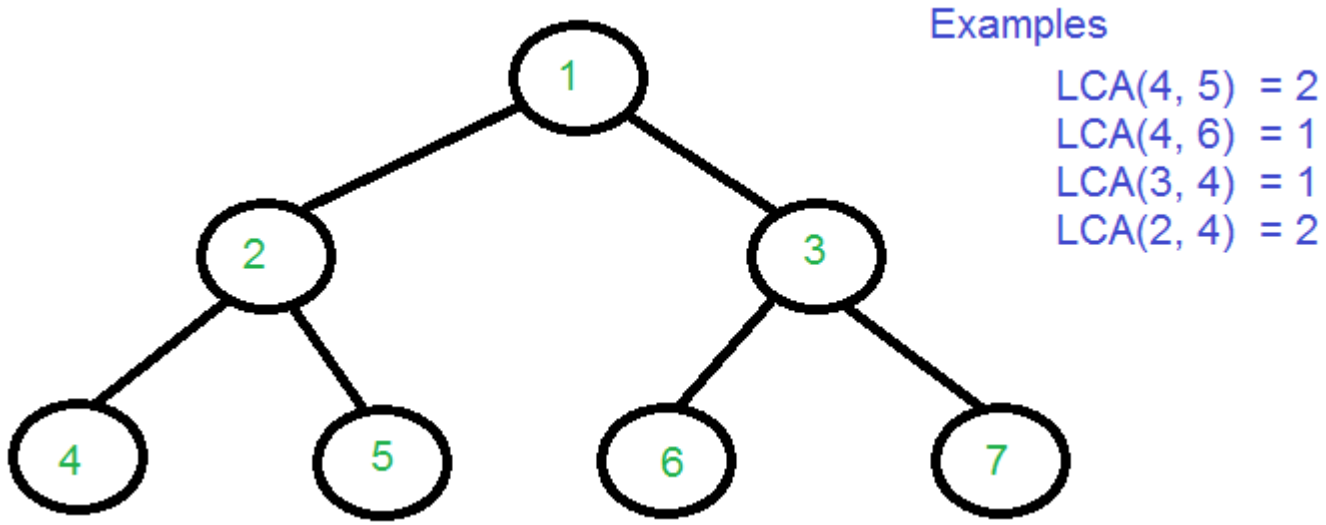
Dash
All
Articles
Videos
Problems
Quiz
Prev
Next

Finding LCA in Binary Tree

Given a **Binary Tree** and the value of two nodes **n1** and **n2**. The task is to find the *lowest common ancestor* of the nodes n1 and n2 in the given Binary Tree.

> *The **LCA** or **Lowest Common Ancestor** of any two nodes N1 and N2 is defined as the common ancestor of both the nodes which is closest to them. That is the distance of the common ancestor from the nodes N1 and N2 should be least possible.*

Below image represents a tree and LCA of different pair of nodes (N1, N2) in it:



Dash
All
Articles
Videos
Problems
Quiz

Finding LCA

Method 1: The simplest method of finding LCA of two nodes in a Binary Tree is to observe that the LCA of the given nodes will be the last common node in the paths from the root node to the given nodes.

For Example: consider the above-given tree and nodes **4** and **5**.

- Path from root to node 4: [1, 2, 4]
- Path from root to node 5: [1, 2, 5].

*The last common node is **2** which will be the LCA.*

Algorithm:

1. Find the path from the **root** node to node **n1** and store it in a vector or array.
2. Find the path from the **root** node to node **n2** and store it in another vector or array.
3. Traverse both paths untill the values in arrays are same. Return the common element just before the mismatch.

Implementation:

C++ Java

```java
// Java Program for Lowest Common Ancestor
// in a Binary Tree

import java.util.ArrayList;
import java.util.List;

// A Binary Tree node
class Node {
    int data;
    Node left, right;

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

public class FindLCA {

    Node root;
    private List<Integer> path1 = new ArrayList<>();
    private List<Integer> path2 = new ArrayList<>();
```

```
// Finds the path from root node to given root of the tree
int findLCA(int n1, int n2)
{
    path1.clear();
    path2.clear();
    return findLCAInternal(root, n1, n2);
}

private int findLCAInternal(Node root, int n1, int n2)
{

    if (!findPath(root, n1, path1) || !findPath(root, n2, path2)) {
        System.out.println((path1.size() > 0) ?
                  "n1 is present" : "n1 is missing");
        System.out.println((path2.size() > 0) ?
                  "n2 is present" : "n2 is missing");
        return -1;
    }

    int i;
    for (i = 0; i < path1.size() && i < path2.size(); i++) {

        if (!path1.get(i).equals(path2.get(i)))
            break;
    }

    return path1.get(i - 1);
}

// Finds the path from root node to given
// root of the tree, Stores the path in a
// vector path[], returns true if path
// exists otherwise false
private boolean findPath(Node root, int n,
                         List<Integer> path)
{
    // base case
    if (root == null) {
        return false;
    }
```

```
        // Store this node. The node will be removed if
        // not in path from root to n.
        path.add(root.data);

        if (root.data == n) {
            return true;
        }

        if (root.left != null && findPath(root.left, n, path)) {
            return true;
        }

        if (root.right != null && findPath(root.right, n, path)) {
            return true;
        }

        // If not present in subtree rooted with root,
        // remove root from path[] and return false
        path.remove(path.size() - 1);

        return false;
    }

    // Driver code
    public static void main(String[] args)
    {
        FindLCA tree = new FindLCA();

        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        System.out.println("LCA(4, 5): " + tree.findLCA(4, 5));
        System.out.println("LCA(4, 6): " + tree.findLCA(4, 6));
        System.out.println("LCA(3, 4): " + tree.findLCA(3, 4));
        System.out.println("LCA(2, 4): " + tree.findLCA(2, 4));
```

```
        }
    }
```


Output:

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

Analysis: *The **time complexity** of the above solution is O(N) where N is the number of nodes in the given Tree and the above solution also takes O(N) **extra space**.*

Method 2: 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 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 the root node. 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 subtrees. The node which has one key present in its left subtree and the other key present in the right subtree is the LCA. If both keys lie in left subtree, then left subtree has LCA also, otherwise, LCA lies in the right subtree.

Below is the implementation of the above approach:

C++ Java


```java
// Java implementation to find lowest common ancestor of
// n1 and n2 using one traversal of 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;
    }
}

// Binary Tree Class
public class BinaryTree {
    // Root of the Binary Tree
    Node root;

    Node findLCA(int n1, int n2)
    {
        return findLCA(root, n1, n2);
    }

    // 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
    Node findLCA(Node node, int n1, int n2)
    {
        if (node == null)
            return null;

        // 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 (node.data == n1 || node.data == n2)
                return node;

            // Look for keys in left and right subtrees
            Node left_lca = findLCA(node.left, n1, n2);
            Node right_lca = findLCA(node.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 != null && right_lca != null)
                return node;

            // Otherwise check if left subtree or right subtree is LCA
            return (left_lca != null) ? left_lca : right_lca;
        }

        // Driver Code
        public static void main(String args[])
        {
            BinaryTree tree = new BinaryTree();
            tree.root = new Node(1);
            tree.root.left = new Node(2);
            tree.root.right = new Node(3);
            tree.root.left.left = new Node(4);
            tree.root.left.right = new Node(5);
            tree.root.right.left = new Node(6);
            tree.root.right.right = new Node(7);

            System.out.println("LCA(4, 5) = " + tree.findLCA(4, 5).data);
            System.out.println("LCA(4, 6) = " + tree.findLCA(4, 6).data);
            System.out.println("LCA(3, 4) = " + tree.findLCA(3, 4).data);
            System.out.println("LCA(2, 4) = " + tree.findLCA(2, 4).data);
        }
    }
```

Dash
All
Articles
Videos
Problems
Quiz

Output:

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