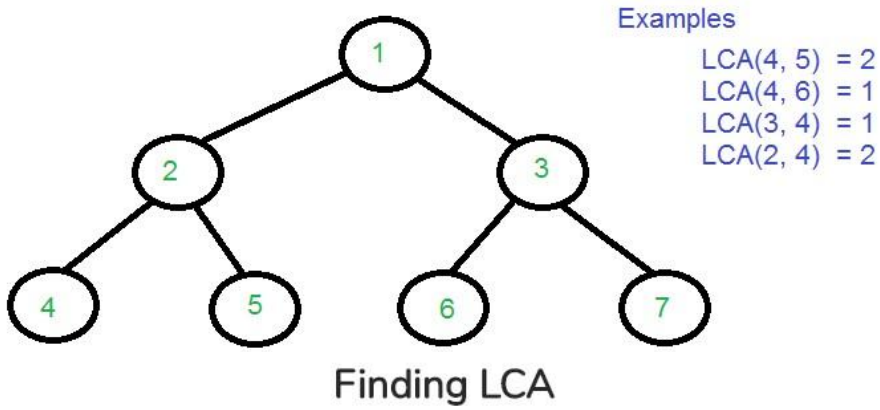
Finding LCA in a Binary Tree

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:



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

```cpp
// C++ Program for Lowest Common Ancestor in a Binary Tree
#include <iostream>
#include <vector>
using namespace std;
// A Binary Tree node
struct Node {
    int key;
    struct Node *left, *right;
};
// Utility function creates a new binary treenode with given key
Node* newNode(int k)
{
    Node* temp = new Node;
    temp->key = k;
    temp->left = temp->right = NULL;
    return temp;
}
// Function to find 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
bool findPath(Node* root, vector<int>& path, int k)
{
    // base case
    if (root == NULL)
        return false;
    // Store this node in path vector The node will be removed if
    // not in path from root to k
    path.push_back(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 && findPath(root->left, path, k)) ||
        (root->right && findPath(root->right, path, k)))
        return true;
    // If not present in subtree rooted with root,
    // remove root from path[] and return false
    path.pop_back();
    return false;
}
// Function to return LCA if node n1, n2 are present in the given binary tree, otherwise
// return -1
int findLCA(Node* root, int n1, int n2)
{
    // to store paths to n1 and n2 from the root
    vector<int> path1, path2;
// Find paths from root to n1 and root to n1.If either n1 or n2 is not present, return -1
    if (!findPath(root, path1, n1) || !findPath(root, path2, n2))
        return -1;
    // Compare the paths to get the first different value
    int i;
```

```
    for (i = 0; i < path1.size() && i < path2.size(); i++)
        if (path1[i] != path2[i])
            break;

    return path1[i - 1];
}

// Driver Code
int main()
{
    // Let us create the Binary Tree
    // as shown in the above diagram
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    cout << "LCA(4, 5) = " << findLCA(root, 4, 5);
    cout << "\nLCA(4, 6) = " << findLCA(root, 4, 6);
    cout << "\nLCA(3, 4) = " << findLCA(root, 3, 4);
    cout << "\nLCA(2, 4) = " << findLCA(root, 2, 4);
    return 0;
}
```

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

```cpp
// C++ Program to find LCA of n1 and n2 usingone traversal of Binary Tree
#include <iostream>
using namespace std;// A Binary Tree Node
struct Node {
    struct Node *left, *right;
    int key;
};
// Utility function to create a new tree Node
Node* newNode(int key)
{
    Node* temp = new Node;
    temp->key = key;
    temp->left = temp->right = NULL;
    return temp;
}
// 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
struct Node* findLCA(struct Node* root, int n1, int n2)
{
    // Base case
    if (root == 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 (root->key == n1 || root->key == n2)
        return root;
    // Look for keys in left and right subtrees
    Node* left_lca = findLCA(root->left, n1, n2);
    Node* 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 && right_lca)
        return root;
    // Otherwise check if left subtree or right subtree is LCA
    return (left_lca != NULL) ? left_lca : right_lca;
}

// Driver Code
int main()
{
    // Let us create binary tree given in the above example
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);

    cout << "LCA(4, 5) = " << findLCA(root, 4, 5)->key;
    cout << "nLCA(4, 6) = " << findLCA(root, 4, 6)->key;
    cout << "nLCA(3, 4) = " << findLCA(root, 3, 4)->key;
    cout << "nLCA(2, 4) = " << findLCA(root, 2, 4)->key;

    return 0;
}
```

Output:

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