Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)

Given values of two nodes in a Binary Tree, find the **L**owest **C**ommon **A**ncestor (LCA). It may be assumed that both nodes exist in the tree.

For example, consider the Binary Tree in diagram, LCA of 10 and 14 is 12 and LCA of 8 and 14 is 8.

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). Source : Wikipedia.

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

We have discussed different approaches to find LCA in set 1. Finding LCA becomes easy when parent pointer is given as we can easily find all ancestors of a node using parent pointer.

Below are steps to find LCA.

1. Create an empty hash table.

Insert n1 and all of its ancestors in hash table.

2. Check if n2 or any of its ancestors exist in hash table, if yes return the first existing ancestor.

Below is C++ implementation of above steps.

```
// C++ program to find lowest common ancestor using parent pointer
#include <bits/stdc++.h>
using namespace std;

// A Tree Node
struct Node
{
    Node *left, *right, *parent;
```

```
    int key;
};

// A utility function to create a new BST node
Node *newNode(int item)
{
    Node *temp = new Node;
    temp->key = item;
    temp->parent = temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with
   given key in Binary Search Tree */
Node *insert(Node *node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
    {
        node->left  = insert(node->left, key);
        node->left->parent = node;
    }
    else if (key > node->key)
    {
        node->right = insert(node->right, key);
        node->right->parent = node;
    }

    /* return the (unchanged) node pointer */
    return node;
}

// To find LCA of nodes n1 and n2 in Binary Tree
Node *LCA(Node *n1, Node *n2)
{
   // Creata a map to store ancestors of n1
   map <Node *, bool> ancestors;

   // Insert n1 and all its ancestors in map
   while (n1 != NULL)
   {
       ancestors[n1] = true;
       n1 = n1->parent;
   }

   // Check if n2 or any of its ancestors is in
   // map.
```

```
    while (n2 != NULL)
    {
        if (ancestors.find(n2) != ancestors.end())
            return n2;
        n2 = n2->parent;
    }

    return NULL;
}

// Driver method to test above functions
int main(void)
{
    Node * root = NULL;

    root = insert(root, 20);
    root = insert(root, 8);
    root = insert(root, 22);
    root = insert(root, 4);
    root = insert(root, 12);
    root = insert(root, 10);
    root = insert(root, 14);

    Node *n1 = root->left->right->left;
    Node *n2 = root->left;
    Node *lca = LCA(n1, n2);

    printf("LCA of %d and %d is %d \n", n1->key, n2->key, lca->key);

    return 0;
}
```

Output:

```
LCA of 10 and 8 is 8
```

Note : The above implementation uses insert of Binary Search Tree to create a Binary Tree, but the function LCA is for any Binary Tree (not necessarily a Binary Search Tree).

Time Complexity : O(h) where h is height of Binary Tree if we use hash table to implement the solution (Note that the above solution uses map which takes O(Log h) time to insert and find). So the time complexity of above implementation is O(h Log h).

Auxiliary Space : O(h)

A O(h) time and O(1) Extra Space Solution

:

The above solution requires extra space because we need to use a hash table to store visited ancestors. We can solve the problem in O(1) extra space using following fact : If both nodes are at same level and if we traverse up using parent pointers of both nodes, the first common node in the path to root is lca.
The idea is to find depths of given nodes and move up the deeper node pointer by the difference between depths. Once both nodes reach same level, traverse them up and return the first common node.

Thanks to Mysterious Mind for suggesting this approach.

```
// C++ program to find lowest common ancestor using parent pointer
#include <bits/stdc++.h>
using namespace std;

// A Tree Node
struct Node
{
    Node *left, *right, *parent;
    int key;
};

// A utility function to create a new BST node
Node *newNode(int item)
{
    Node *temp = new Node;
    temp->key = item;
    temp->parent = temp->left = temp->right = NULL;
    return temp;
}

/* A utility function to insert a new node with
given key in Binary Search Tree */
Node *insert(Node *node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
    {
        node->left = insert(node->left, key);
        node->left->parent = node;
    }
    else if (key > node->key)
    {
        node->right = insert(node->right, key);
        node->right->parent = node;
    }

    /* return the (unchanged) node pointer */
```

```
    return node;
}

// A utility function to find depth of a node
// (distance of it from root)
int depth(Node *node)
{
    int d = -1;
    while (node)
    {
        ++d;
        node = node->parent;
    }
    return d;
}

// To find LCA of nodes n1 and n2 in Binary Tree
Node *LCA(Node *n1, Node *n2)
{
    // Find depths of two nodes and differences
    int d1 = depth(n1), d2 = depth(n2);
    int diff = d1 - d2;

    // If n2 is deeper, swap n1 and n2
    if (diff < 0)
    {
        Node * temp = n1;
        n1 = n2;
        n2 = temp;
        diff = -diff;
    }

    // Move n1 up until it reaches the same level as n2
    while (diff--)
        n1 = n1->parent;

    // Now n1 and n2 are at same levels
    while (n1 && n2)
    {
        if (n1 == n2)
            return n1;
        n1 = n1->parent;
        n2 = n2->parent;
    }

    return NULL;
}

// Driver method to test above functions
int main(void)
```

```
{
    Node * root = NULL;

    root = insert(root, 20);
    root = insert(root, 8);
    root = insert(root, 22);
    root = insert(root, 4);
    root = insert(root, 12);
    root = insert(root, 10);
    root = insert(root, 14);

    Node *n1 = root->left->right->left;
    Node *n2 = root->right;

    Node *lca = LCA(n1, n2);
    printf("LCA of %d and %d is %d \n", n1->key, n2->key, lca->key);

    return 0;
}
```

Output :

```
LCA of 10 and 22 is 20
```

You may like to see below articles as well :

Lowest Common Ancestor in a Binary Tree | Set 1

Lowest Common Ancestor in a Binary Search Tree.

Find LCA in Binary Tree using RMQ

This article is contributed by **Dheeraj Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Leave a comment Category: Trees

Related Posts:

- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?
- Closest leaf to a given node in Binary Tree
- Locking and Unlocking of Resources arranged in the form of n-ary Tree
- Find all possible binary trees with given Inorder Traversal

(Login to Rate and Mark)

2 Average Difficulty : **2/5.0**
Based on **4** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.