Binary Tree Diameter (Medium)

Given a binary tree in which each node contains an integer number. The diameter is defined as the longest distance from one leaf node to another leaf node. The distance is the number of nodes on the path.
If there does not exist any such paths, return 0.

Examples

```
   5
 /   \
2     11
     /   \
    6     14
```

The diameter of this tree is 4 (2 → 5 → 11 → 14)

How is the binary tree represented?
We use the level order traversal sequence with a special symbol "#" denoting the null node.

For Example:
The sequence [1, 2, 3, #, #, 4] represents the following binary tree:

```
   1
 /   \
2     3
     /
    4
```

```cpp
int Diameter(vector<char> tree, int root, int& global_max){
    if(root > tree.size() - 1 || tree[root] == '#')
        return 0;
    int left_height = Diameter(tree, 2 * root + 1, global_max);
    int right_height = Diameter(tree, 2 * root + 2, global_max);
    global_max = max(global_max, left_height + right_height + 1 );
    return max(left_height, right_height) + 1;
}
```

Time Complexity: O(n), since we traverse the tree once.

正确！Accepted！

Count Univalue Subtrees (Medium)

Given a binary tree, count the number of uni-value subtrees.
A Uni-value subtree means all nodes of the subtree have the same value.

For example:
Given binary tree,

```
      5
     / \
    1   5
   / \   \
  5   5   5
```

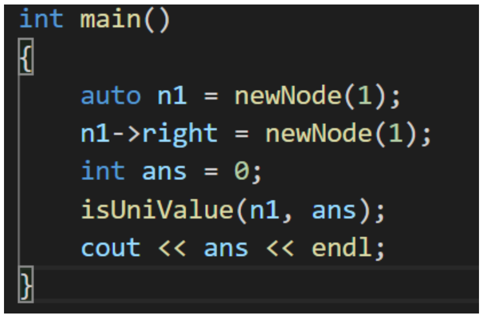
return 4.

```
struct Node{
    int data;
    Node* left, *right;
};

struct Node* newNode(int data){
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

bool isUniValue(Node* root, int& count){
    if(root == NULL)
        return false;
    if(root->left == NULL && root->right == NULL){
        count++;
        return true;
    }
    bool left_isUniVal = isUniValue(root->left, count);
    bool right_isUniVal = isUniValue(root->right, count);
    if(left_isUniVal && right_isUniVal && root->left->data == root->right->data) {
        count++;
        return true;
    }
    else
        return false;
}
```

Time Complexity: O(n), since we traverse the tree once.

# 答案错误！Wrong Answer！

# 若某个根只有左子树或右子树，且为 Univalue Subtrees，会被漏判



```
int main()
{
    auto n1 = newNode(1);
    n1->right = newNode(1);
    int ans = 0;
    isUniValue(n1, ans);
    cout << ans << endl;
}
```

（ans 应该为 2）

Lowest Common Ancestor I(Medium)

Given two nodes in a binary tree, find their lowest common ancestor.

Assumptions

- There is no parent pointer for the nodes in the binary tree
- The given two nodes are guaranteed to be in the binary tree

Examples

```
        5
      /   \
     9     12
   /  \      \
  2    3      14
```

The lowest common ancestor of 2 and 14 is 5
The lowest common ancestor of 2 and 9 is 9

```
struct Node{
    int data;
    Node* left, *right;
};

struct Node* newNode(int data){
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* findLCA1(Node* root, int a, int b){
    if(root == NULL || root->data == a || root->data == b)
        return root;
    Node* leftLCA = findLCA1(root->left, a, b);
    Node* rightLCA = findLCA1(root->right, a, b);
    if(leftLCA && rightLCA)
        return root;
    else
        return (leftLCA == NULL) ? rightLCA : leftLCA;
}
```

Time Complexity: O(n), since we traverse the tree once.

# 正确！Accepted！

Lowest Common Ancestor II (Medium)

Given two nodes in a binary tree (with parent pointer available), find their lowest common ancestor.

Assumptions

- There is parent pointer for the nodes in the binary tree
- The given two nodes are not guaranteed to be in the binary tree

Examples

```
      5
     /  \
    9    12
  / \     \
 2   3     14
```

The lowest common ancestor of 2 and 14 is 5
The lowest common ancestor of 2 and 9 is 9
The lowest common ancestor of 2 and 8 is null (8 is not in the tree)

```cpp
struct Node{
    int data;
    Node* left, *right;
};

struct Node* newNode(int data){
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* findLCA1(Node* root, int a, int b){
    if(root == NULL || root->data == a || root->data == b)
        return root;
    Node* leftLCA = findLCA1(root->left, a, b);
    Node* rightLCA = findLCA1(root->right, a, b);
    if(leftLCA && rightLCA)
        return root;
    else
        return (leftLCA == NULL) ? rightLCA : leftLCA;
}

bool isTreeNode(Node* root, int a){
    if(root == NULL)
        return false;
    if(root->data == a)
        return true;
    bool InLeft = isTreeNode(root->left, a);
    bool InRight = isTreeNode(root->right, a);
    return InLeft || InRight;
}

struct Node* findLCA2(Node* root, int a, int b){
    if(isTreeNode(root, a) && isTreeNode(root, b))
        return findLCA1(root, a, b);
    else
        return NULL;
}
```

Time Complexity: O(n), since we traverse the tree three times.

# 正确！Accepted！

# 本题的节点可以拥有父节点指针，请思考能不能利用父节点指针使得不需要第二次遍历整颗树来寻找 LCA？

Distance Of Two Nodes In Binary Tree (Medium)

Find distance between two given keys of a Binary Tree, no parent pointers are given. Distance between two nodes is the minimum number of edges to be traversed to reach one node from other.

Assumptions:

- There are no duplicate keys in the binary tree.
- The given two keys are guaranteed to be in the binary tree.
- The given two keys are different.

Examples:

```
    1
   / \
  2   3
 / \ / \
4  5 6  7
     \
      8
```

distance(4, 5) = 2
distance(4, 6) = 4

```
struct Node{
    int data;
    Node* left, *right;
};

struct Node* newNode(int data){
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* findLCA1(Node* root, int a, int b){
    if(root == NULL || root->data == a || root->data == b)
        return root;
    Node* leftLCA = findLCA1(root->left, a, b);
    Node* rightLCA = findLCA1(root->right, a, b);
    if(leftLCA && rightLCA)
        return root;
    else
        return (leftLCA == NULL) ? rightLCA : leftLCA;
}
```

```
int distanceToLCA(Node* root, int a, int dist){
    if(root == NULL) return -1;
    if(root->data == a) return dist;
    int left = distanceToLCA(root->left, a, dist+1);
    if(left == -1)
        return distanceToLCA(root->right, a, dist+1);
    return left;
}

int distance(Node*root, int a, int b){
    struct Node* rootLCA = findLCA1(root, a, b);
    cout << rootLCA->data << "," << distanceToLCA(rootLCA, a, 0) << "," <<
distanceToLCA(rootLCA, b, 0) << endl;
    return distanceToLCA(rootLCA, a, 0) + distanceToLCA(rootLCA, b, 0);
}
```

Time Complexity: O(n).

# 正确！Accepted！

Binary Tree Paths (Easy)

Given a binary tree, return all root-to-leaf paths **from left to right**.

For example, given the following binary tree:

```
   1
 /   \
2     3
 \
  5
```

All root-to-leaf paths are:
`["1->2->5", "1->3"]`

```
struct Node{
    int data;
    Node* left, *right;
};

struct Node* newNode(int data){
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

void binaryTreePath(Node* root, vector<string>& paths, string path){
    path += to_string(root->data);
    if(!root->left && !root->right) {
        paths.push_back(path);
        return;
    }
    if(root->left)
        binaryTreePath(root->left, paths, path + "->");
    if(root->right)
        binaryTreePath(root->right, paths, path + "->");
}

vector<string> binaryTreePath(Node* root, vector<string>& paths){
    binaryTreePath(root, paths, "");
    return paths;
}
```

Time Complexity: O(n).

```
# 正确！Accepted！
```