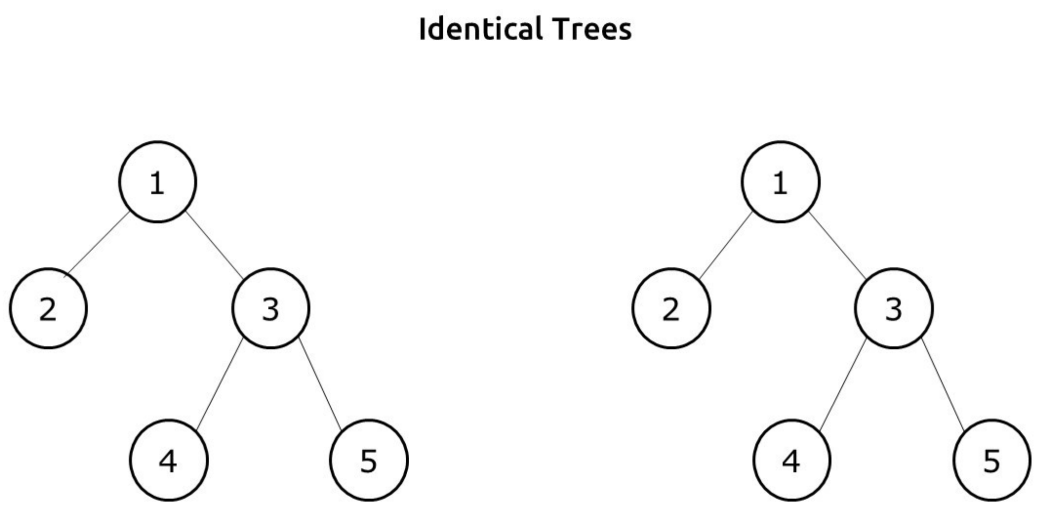
Check if two trees are identical
Problem Statement: Given two Binary Tree. Write a program to check if two trees are identical or not.
Example 1:

Input:

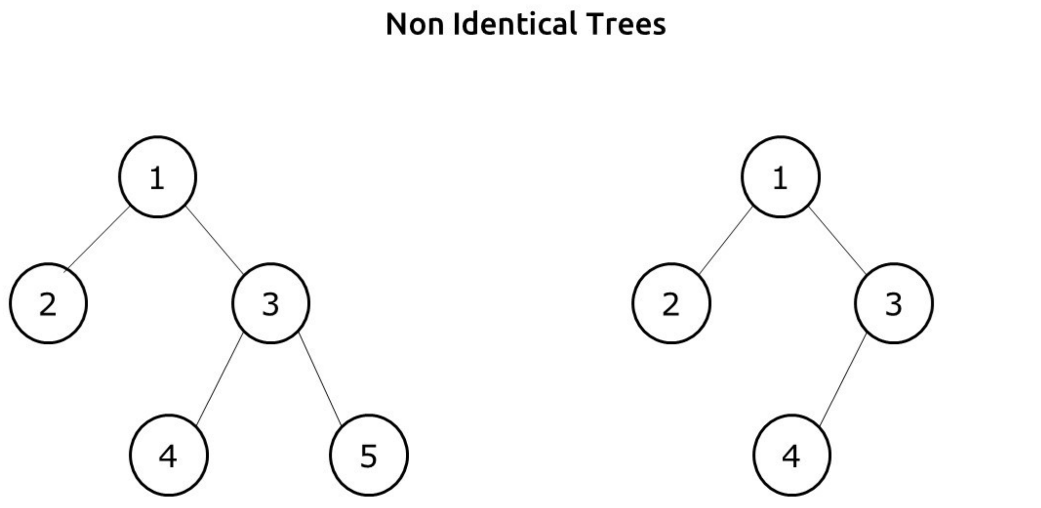


Output: Two Trees are identical

Exaplaination: Two trees T1 and T2 are said to be identical if these three conditions are met for every pair of nodes :
Value of a node in T1 is equal to the value of the corresponding node in T2.
Left subtree of this node is identical to the left subtree of the corresponding node.
Right subtree of this node is identical to the right subtree of the corresponding node.

Example 2:

Input:

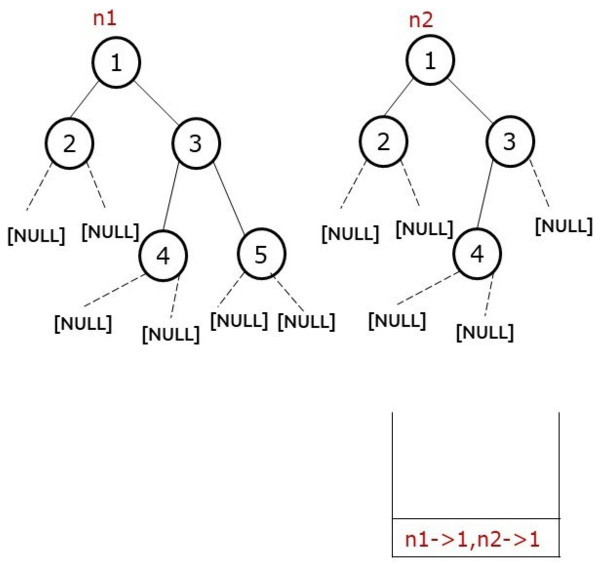


Output: Two Trees are not identical

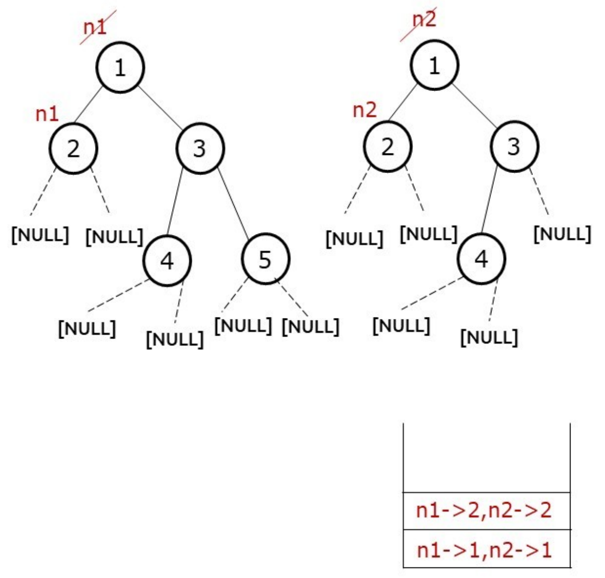
__Disclaimer__: Don't jump directly to the solution, try it out yourself first.

Solution:

Approach: In order to check whether two trees are identical or not, we need to traverse the trees. While traversing we first check the value of the nodes, if they are unequal we can simply return false, as trees are non-identical. If they are the same, then we need to **recursively** check their left child as well as the right child. When we get all the three values as true(node values, left child, right child) we can conclude that these are identical trees and can return true. Any other combination will return false.
Dry Run:

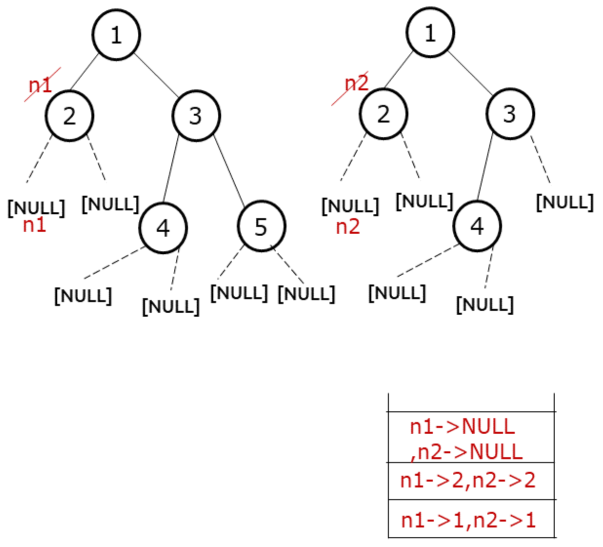


(i)

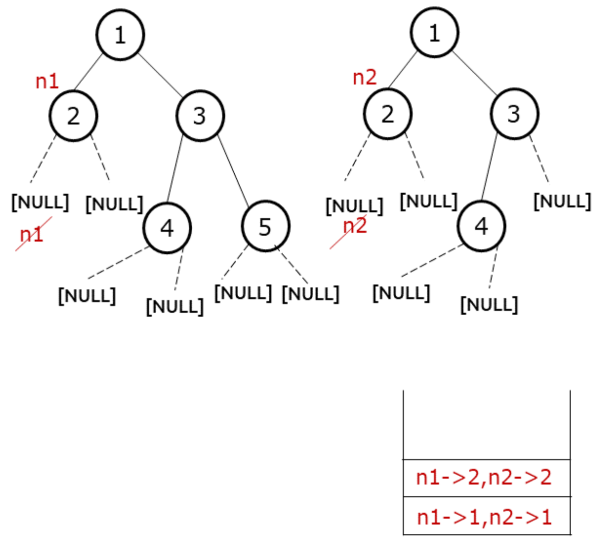


(ii)

(i) Initially we have variables n1 and n2 pointing to the roots of both trees. This function is inside our recursion stack as well.
(ii) First we check the values at the nodes. As the values of nodes are equal, we proceed further and recursively call the left child of both nodes. This second function is pushed to our recursion stack.

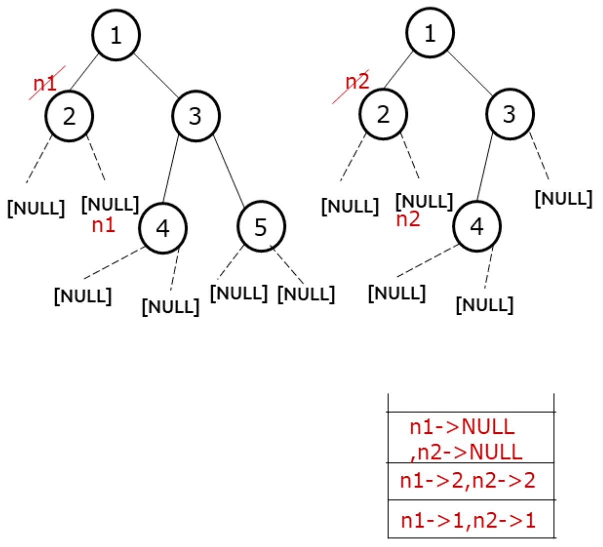


(iii)

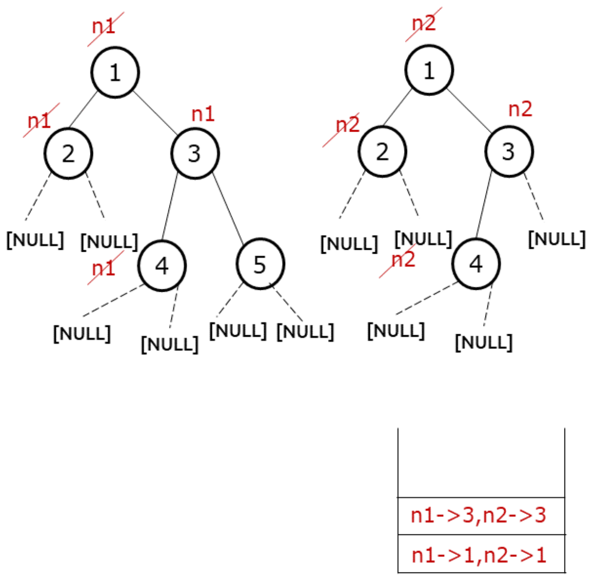


(iv)

(iii) Again we check the nodes. As they are equal, we again check for the left subtrees and add the function to our recursion stack.
(**iv)** Both n1 and n2 point to NULL, so the base condition will be true and we will return true from this function.

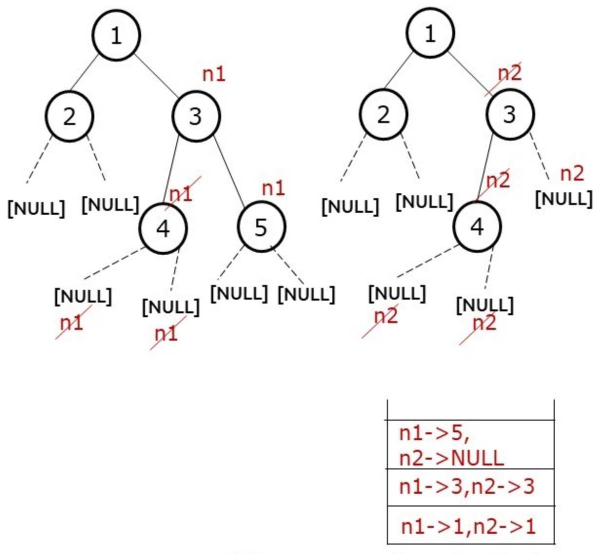


(v) Recursion Stack

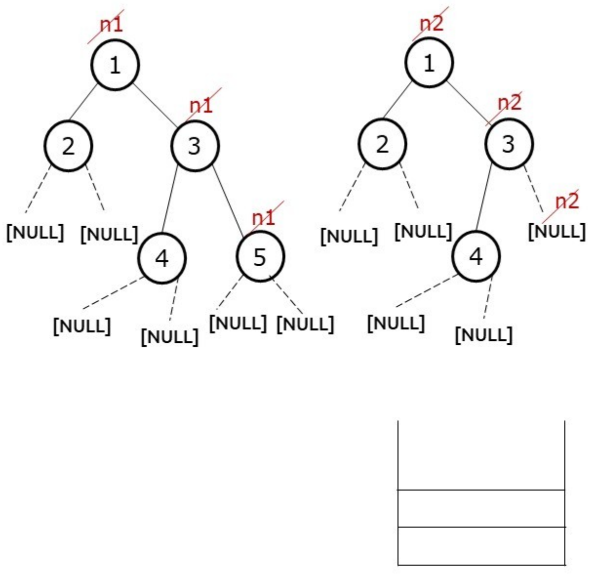


(vi) Recursion Stack

(v) At nodes pointing to 2, two out of three conditions return true, so we will proceed further and check the third condition. We will recursively call the right child of the nodes. Again that function will be added to our recursive stack.
(vi) Similar to step 4, n1 and n2 point to NULL, therefore we will again return true and the function will be removed from our recursion stack. Now at nodes pointing to 2, all three conditions return true (node values, left child and right child), therefore we can return true from this function and remove it from the recursion stack. Then at node 1, two conditions are true, therefore we call the right child of n1 and n2 to check the third condition.



(vii) Recursion Stack



(viii) Recursion Stack

(vii) At nodes pointing to 3, we first check the data values which are equal, then we recursively call the function to check for their left child. It turns out to be identical and we get the return value true, then we recursively call the function to check for their right child.
(viii) Now n1 points to a node with value 5 whereas n2 points to NULL, this can't be the case with an identical tree, our second base condition hits and we return false from this function. In the parent function where n1 and n2 point to 3, **only two out of three conditions return true,** therefore we will return the value false. In its parent function where n1 and n2 points to 1, again **only two out of three conditions return true,** therefore we will return false. Hence we will return a false value.
As our first function returns a false to our main function, we can conclude that these two trees are **Non-identical.**
Code:

- C++ Code
- Java Code

```
#include <bits/stdc++.h>

using namespace std;
```

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

bool isIdentical(node * node1, node * node2) {
   if (node1 == NULL && node2 == NULL)
      return true;
   else if (node1 == NULL || node2 == NULL)
      return false;

   return ((node1 -> data == node2 -> data) && isIdentical(node1 -> left, node2 -> left) && isIdentical(node1 -> right, node2 -> right));
}

struct node * newNode(int data) {
   struct node * node = (struct node * ) malloc(sizeof(struct node));
   node -> data = data;
   node -> left = NULL;
   node -> right = NULL;

   return (node);
}

int main() {

   struct node * root1 = newNode(1);
   root1 -> left = newNode(2);
   root1 -> right = newNode(3);
   root1 -> right -> left = newNode(4);
   root1 -> right -> right = newNode(5);

   struct node * root2 = newNode(1);
   root2 -> left = newNode(2);
   root2 -> right = newNode(3);
   root2 -> right -> left = newNode(4);

   if (isIdentical(root1, root2))
      cout << "Two Trees are identical";
   else cout << "Two trees are non-identical";

   return 0;
}
```

Output:

```
Two trees are non-identical
```

Time Complexity: O(N).

Reason: We are doing a tree traversal.

Space Complexity: O(N)

Reason: Space is needed for the recursion stack. In the worst case (skewed tree), space complexity can be O(N).