

Problem 100: Same Tree

Problem Information

Difficulty: **Easy**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the roots of two binary trees

p

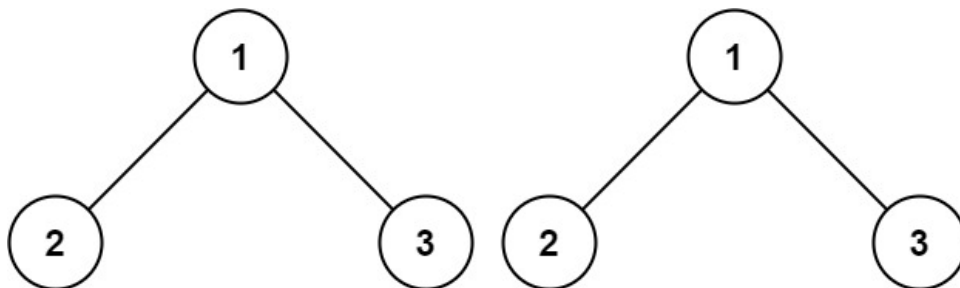
and

q

, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:



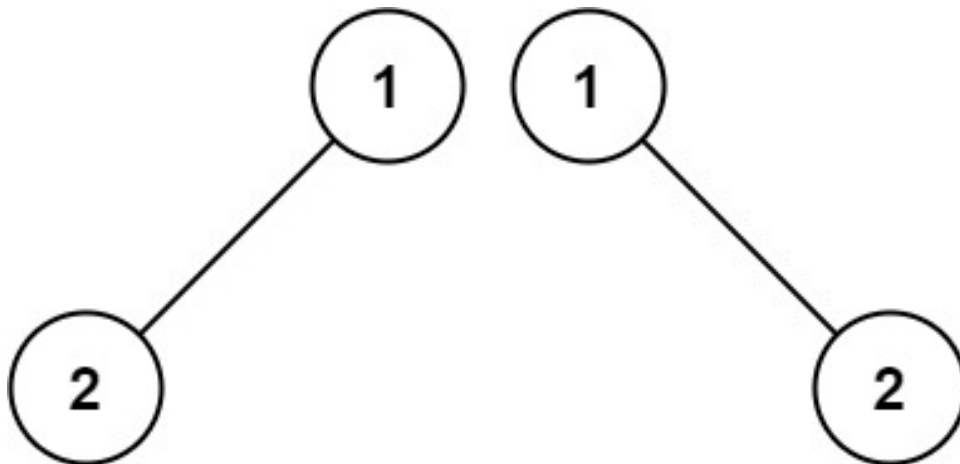
Input:

$p = [1,2,3]$, $q = [1,2,3]$

Output:

true

Example 2:



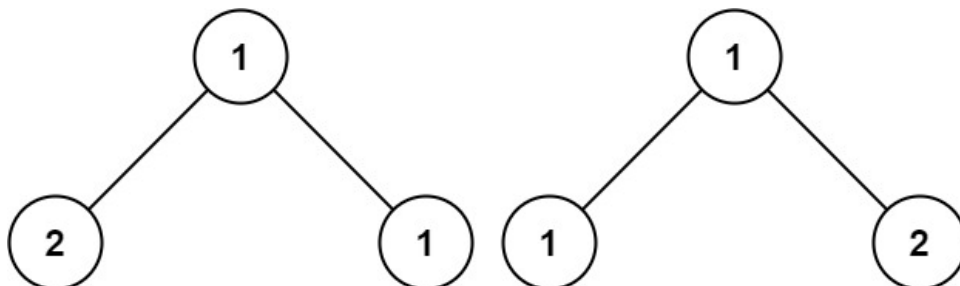
Input:

$p = [1,2]$, $q = [1,\text{null},2]$

Output:

false

Example 3:



Input:

$p = [1,2,1]$, $q = [1,1,2]$

Output:

false

Constraints:

The number of nodes in both trees is in the range

[0, 100]

.

-10

4

$\leq \text{Node.val} \leq 10$

4

Code Snippets

C++:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *   int val;
 *   TreeNode *left;
 *   TreeNode *right;
 *   TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *   TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {

    }
```

```
};
```

Java:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *   int val;
 *   TreeNode left;
 *   TreeNode right;
 *   TreeNode() {}
 *   TreeNode(int val) { this.val = val; }
 *   TreeNode(int val, TreeNode left, TreeNode right) {
 *     this.val = val;
 *     this.left = left;
 *     this.right = right;
 *   }
 * }
 */
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {

    }
}
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:
```

Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
```

```

# self.left = left
# self.right = right
class Solution(object):
def isSameTree(self, p, q):
    """
    :type p: Optional[TreeNode]
    :type q: Optional[TreeNode]
    :rtype: bool
    """

```

JavaScript:

```

/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *   this.val = (val===undefined ? 0 : val)
 *   this.left = (left===undefined ? null : left)
 *   this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} p
 * @param {TreeNode} q
 * @return {boolean}
 */
var isSameTree = function(p, q) {

};

```

TypeScript:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 *   val: number
 *   left: TreeNode | null
 *   right: TreeNode | null
 *   constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
 *   {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 *   }
 * }

```

```

* }
* }
*/

function isSameTree(p: TreeNode | null, q: TreeNode | null): boolean {

};

```

C#:

```

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public bool IsSameTree(TreeNode p, TreeNode q) {

    }
}

```

C:

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
bool isSameTree(struct TreeNode* p, struct TreeNode* q) {

}

```

Go:

```
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func isSameTree(p *TreeNode, q *TreeNode) bool {

}
```

Kotlin:

```
/**
 * Example:
 * var ti = TreeNode(5)
 * var v = ti.`val`
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
 *     var left: TreeNode? = null
 *     var right: TreeNode? = null
 * }
 */
class Solution {
    fun isSameTree(p: TreeNode?, q: TreeNode?): Boolean {

    }
}
```

Swift:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public var val: Int
 *     public var left: TreeNode?
 *     public var right: TreeNode?
 *     public init() { self.val = 0; self.left = nil; self.right = nil; }
 *     public init(_ val: Int) { self.val = val; self.left = nil; self.right = nil; }
 * }
 */
```

```

* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
*   self.val = val
*   self.left = left
*   self.right = right
* }
* }
*/
class Solution {
func isSameTree(_ p: TreeNode?, _ q: TreeNode?) -> Bool {

}
}

```

Rust:

```

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
//   pub val: i32,
//   pub left: Option<Rc<RefCell<TreeNode>>>,
//   pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
//   #[inline]
//   pub fn new(val: i32) -> Self {
//     TreeNode {
//       val,
//       left: None,
//       right: None
//     }
//   }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn is_same_tree(p: Option<Rc<RefCell<TreeNode>>>, q:
Option<Rc<RefCell<TreeNode>>>) -> bool {

}
}

```


Ruby:

```
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end

# @param {TreeNode} p
# @param {TreeNode} q
# @return {Boolean}
def is_same_tree(p, q)

end
```

PHP:

```
/**
 * Definition for a binary tree node.
 * class TreeNode {
 * public $val = null;
 * public $left = null;
 * public $right = null;
 * function __construct($val = 0, $left = null, $right = null) {
 * $this->val = $val;
 * $this->left = $left;
 * $this->right = $right;
 * }
 * }
 */

class Solution {

/**
 * @param TreeNode $p
 * @param TreeNode $q
 * @return Boolean
 */
function isSameTree($p, $q) {
```

```
}  
}
```

Dart:

```
/**  
 * Definition for a binary tree node.  
 * class TreeNode {  
 *   int val;  
 *   TreeNode? left;  
 *   TreeNode? right;  
 *   TreeNode([this.val = 0, this.left, this.right]);  
 * }  
 */  
class Solution {  
  bool isSameTree(TreeNode? p, TreeNode? q) {  
  
  }  
}
```

Scala:

```
/**  
 * Definition for a binary tree node.  
 * class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =  
 * null) {  
 *   var value: Int = _value  
 *   var left: TreeNode = _left  
 *   var right: TreeNode = _right  
 * }  
 */  
object Solution {  
  def isSameTree(p: TreeNode, q: TreeNode): Boolean = {  
  
  }  
}
```

Elixir:

```
# Definition for a binary tree node.  
#  
# defmodule TreeNode do
```

```

# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec is_same_tree(p :: TreeNode.t | nil, q :: TreeNode.t | nil) :: boolean
def is_same_tree(p, q) do

end
end

```

Erlang:

```

%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec is_same_tree(P :: #tree_node{} | null, Q :: #tree_node{} | null) ->
boolean().
is_same_tree(P, Q) ->
.

```

Racket:

```

; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

```

```
|#

(define/contract (is-same-tree p q)
  (-> (or/c tree-node? #f) (or/c tree-node? #f) boolean?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Same Tree
 * Difficulty: Easy
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *   int val;
 *   TreeNode *left;
 *   TreeNode *right;
 *   TreeNode() : val(0), left(nullptr), right(nullptr) {
 * // TODO: Implement optimized solution
 *   return 0;
 * }
 *   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {
 * // TODO: Implement optimized solution
 *   return 0;
 * }
 *   TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
 *   right(right) {
 * // TODO: Implement optimized solution
 *   return 0;
 * }
 */
```

```

* };
*/
class Solution {
public:
bool isSameTree(TreeNode* p, TreeNode* q) {

}
};

```

Java Solution:

```

/**
 * Problem: Same Tree
 * Difficulty: Easy
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {
 * // TODO: Implement optimized solution
 *     return 0;
 * }
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */

class Solution {
public boolean isSameTree(TreeNode p, TreeNode q) {

```

```
}  
}
```

Python3 Solution:

```
"""  
Problem: Same Tree  
Difficulty: Easy  
Tags: tree, search  
  
Approach: DFS or BFS traversal  
Time Complexity: O(n) where n is number of nodes  
Space Complexity: O(h) for recursion stack where h is height  
"""  
  
# Definition for a binary tree node.  
# class TreeNode:  
#     def __init__(self, val=0, left=None, right=None):  
#         self.val = val  
#         self.left = left  
#         self.right = right  
class Solution:  
    def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

```
# Definition for a binary tree node.  
# class TreeNode(object):  
#     def __init__(self, val=0, left=None, right=None):  
#         self.val = val  
#         self.left = left  
#         self.right = right  
class Solution(object):  
    def isSameTree(self, p, q):  
        """  
        :type p: Optional[TreeNode]  
        :type q: Optional[TreeNode]  
        :rtype: bool  
        """
```

```
"""
```

JavaScript Solution:

```
/**
 * Problem: Same Tree
 * Difficulty: Easy
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *   this.val = (val===undefined ? 0 : val)
 *   this.left = (left===undefined ? null : left)
 *   this.right = (right===undefined ? null : right)
 * }
 */

/**
 * @param {TreeNode} p
 * @param {TreeNode} q
 * @return {boolean}
 */
var isSameTree = function(p, q) {

};
```

TypeScript Solution:

```
/**
 * Problem: Same Tree
 * Difficulty: Easy
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
```

```

*/

/**
 * Definition for a binary tree node.
 * class TreeNode {
 *   val: number
 *   left: TreeNode | null
 *   right: TreeNode | null
 *   constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
 *   {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 *   }
 * }
 */

function isSameTree(p: TreeNode | null, q: TreeNode | null): boolean {

};

```

C# Solution:

```

/*
 * Problem: Same Tree
 * Difficulty: Easy
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *   public int val;
 *   public TreeNode left;
 *   public TreeNode right;
 *   public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *     this.val = val;

```



```

    * this.left = left;
    * this.right = right;
    * }
    * }
    */
public class Solution {
    public bool IsSameTree(TreeNode p, TreeNode q) {

    }
}

```

C Solution:

```

/*
 * Problem: Same Tree
 * Difficulty: Easy
 * Tags: tree, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
bool isSameTree(struct TreeNode* p, struct TreeNode* q) {

}

```

Go Solution:

```

// Problem: Same Tree
// Difficulty: Easy
// Tags: tree, search
//

```

```

// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func isSameTree(p *TreeNode, q *TreeNode) bool {

}

```

Kotlin Solution:

```

/**
 * Example:
 * var ti = TreeNode(5)
 * var v = ti.`val`
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
 *     var left: TreeNode? = null
 *     var right: TreeNode? = null
 * }
 */
class Solution {
    fun isSameTree(p: TreeNode?, q: TreeNode?): Boolean {

    }
}

```

Swift Solution:

```

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public var val: Int
 *     public var left: TreeNode?

```

```

* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func isSameTree(_ p: TreeNode?, _ q: TreeNode?) -> Bool {

}
}

```

Rust Solution:

```

// Problem: Same Tree
// Difficulty: Easy
// Tags: tree, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
//     pub val: i32,
//     pub left: Option<Rc<RefCell<TreeNode>>>,
//     pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
//     #[inline]
//     pub fn new(val: i32) -> Self {
//         TreeNode {
//             val,
//             left: None,

```

```

// right: None
// }
// }
// }

use std::rc::Rc;
use std::cell::RefCell;

impl Solution {
    pub fn is_same_tree(p: Option<Rc<RefCell<TreeNode>>>, q:
Option<Rc<RefCell<TreeNode>>>) -> bool {

    }
}

```

Ruby Solution:

```

# Definition for a binary tree node.
# class TreeNode
#   attr_accessor :val, :left, :right
#   def initialize(val = 0, left = nil, right = nil)
#     @val = val
#     @left = left
#     @right = right
#   end
# end

# @param {TreeNode} p
# @param {TreeNode} q
# @return {Boolean}

def is_same_tree(p, q)

end

```

PHP Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 *   public $val = null;
 *   public $left = null;
 *   public $right = null;
 *   function __construct($val = 0, $left = null, $right = null) {
 *     $this->val = $val;

```

```

* $this->left = $left;
* $this->right = $right;
* }
* }
*/
class Solution {

/**
 * @param TreeNode $p
 * @param TreeNode $q
 * @return Boolean
 */
function isSameTree($p, $q) {

}

}

```

Dart Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 *   int val;
 *   TreeNode? left;
 *   TreeNode? right;
 *   TreeNode([this.val = 0, this.left, this.right]);
 * }
 */
class Solution {
  bool isSameTree(TreeNode? p, TreeNode? q) {

  }

}

```

Scala Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
 *   var value: Int = _value

```

```

* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def isSameTree(p: TreeNode, q: TreeNode): Boolean = {

}
}

```

Elixir Solution:

```

# Definition for a binary tree node.
#
# defmodule TreeNode do
#   @type t :: %__MODULE__{
#     val: integer,
#     left: TreeNode.t() | nil,
#     right: TreeNode.t() | nil
#   }
#   defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
  @spec is_same_tree(p :: TreeNode.t | nil, q :: TreeNode.t | nil) :: boolean
  def is_same_tree(p, q) do

  end

end

```

Erlang Solution:

```

%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{}},
%% right = null :: 'null' | #tree_node{}}).

-spec is_same_tree(P :: #tree_node{} | null, Q :: #tree_node{} | null) ->
boolean().
is_same_tree(P, Q) ->

```

.

Racket Solution:

```
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
  (val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
  (tree-node val #f #f))

|#

(define/contract (is-same-tree p q)
  (-> (or/c tree-node? #f) (or/c tree-node? #f) boolean?)
  )
```