# Problem 993: Cousins in Binary Tree

## Problem Information

**Difficulty:** Easy
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

Given the

root

of a binary tree with unique values and the values of two different nodes of the tree

x

and

y

, return

true

if the nodes corresponding to the values

x

and

y

in the tree are

cousins

, or

false

otherwise.

Two nodes of a binary tree are

cousins

if they have the same depth with different parents.

Note that in a binary tree, the root node is at the depth

0

, and children of each depth

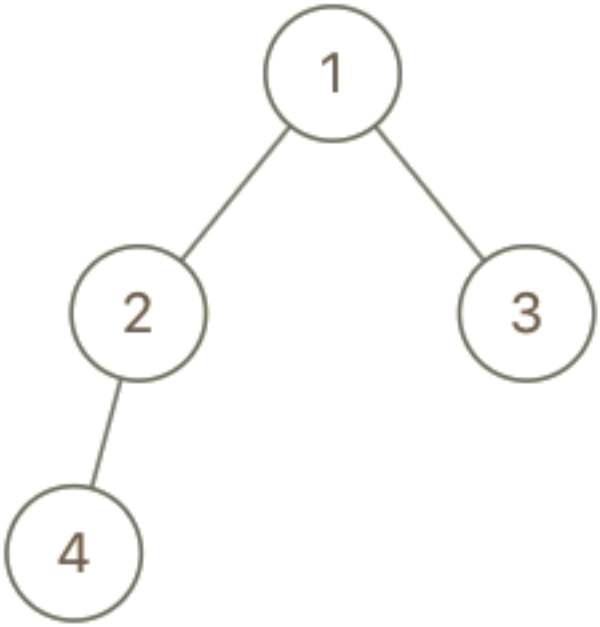
k

node are at the depth

k + 1

.

Example 1:



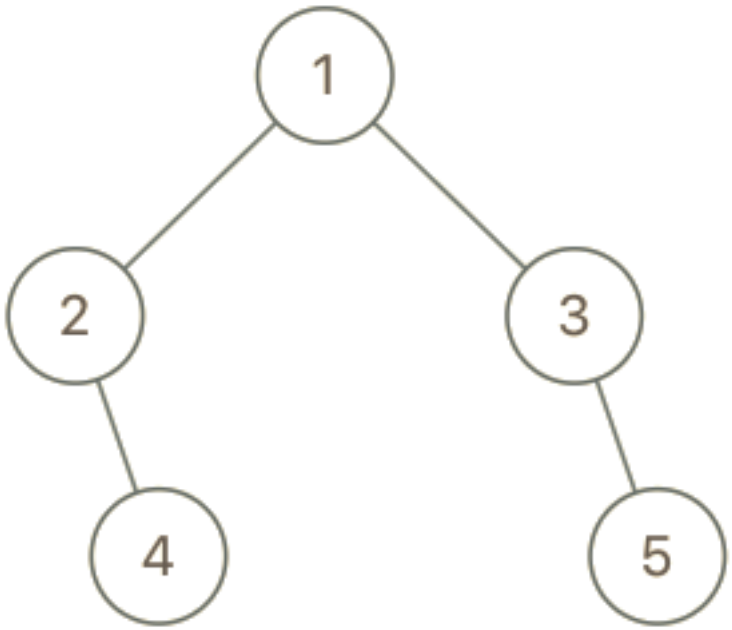

Input:

root = [1,2,3,4], x = 4, y = 3

Output:

false

Example 2:



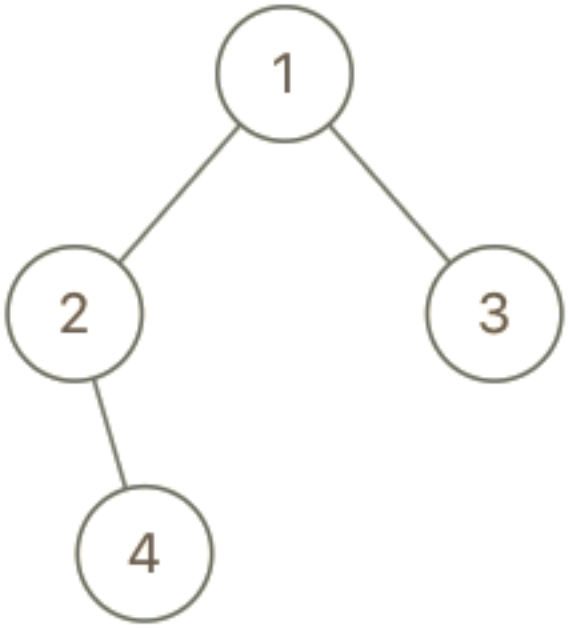

Input:

root = [1,2,3,null,4,null,5], x = 5, y = 4

Output:

true

Example 3:




Input:

root = [1,2,3,null,4], x = 2, y = 3

Output:

false

Constraints:

The number of nodes in the tree is in the range

[2, 100]

.

1 <= Node.val <= 100

Each node has a

unique

value.

x != y

x

and

y

are exist in the tree.

## 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 isCousins(TreeNode* root, int x, int y) {

}
};
```

**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 isCousins(TreeNode root, int x, int y) {

}
}
```

**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 isCousins(self, root: Optional[TreeNode], x: int, y: int) -> 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 isCousins(self, root, x, y):
"""
:type root: Optional[TreeNode]
:type x: int
:type y: int
: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} root
* @param {number} x
* @param {number} y
* @return {boolean}
*/
var isCousins = function(root, x, y) {

};
```

**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 isCousins(root: TreeNode | null, x: number, y: number): 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 IsCousins(TreeNode root, int x, int y) {

}
}
```

**C:**

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
bool isCousins(struct TreeNode* root, int x, int y) {

}
```

**Go:**

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
```

```
* }
*/
func isCousins(root *TreeNode, x int, y int) 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 isCousins(root: TreeNode?, x: Int, y: Int): 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 isCousins(_ root: TreeNode?, _ x: Int, _ y: Int) -> 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_cousins(root: Option<Rc<RefCell<TreeNode>>>, x: i32, y: i32) ->
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} root
# @param {Integer} x
# @param {Integer} y
# @return {Boolean}
def is_cousins(root, x, y)

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 $root
* @param Integer $x
* @param Integer $y
* @return Boolean
*/
function isCousins($root, $x, $y) {

}
}
```

**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 isCousins(TreeNode? root, int x, int y) {

}
}
```

**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 isCousins(root: TreeNode, x: Int, y: Int): 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_cousins(root :: TreeNode.t | nil, x :: integer, y :: integer) ::
boolean
def is_cousins(root, x, y) 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_cousins(Root :: #tree_node{} | null, X :: integer(), Y :: integer())
-> boolean().
is_cousins(Root, X, Y) ->
.
```

**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-cousins root x y)
(-> (or/c tree-node? #f) exact-integer? exact-integer? boolean?)
```

```
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Cousins in Binary 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 isCousins(TreeNode* root, int x, int y) {
```

```
}
};
```

**Java Solution:**

```
/**
* Problem: Cousins in Binary 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 isCousins(TreeNode root, int x, int y) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Cousins in Binary 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 isCousins(self, root: Optional[TreeNode], x: int, y: int) -> 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 isCousins(self, root, x, y):
"""
:type root: Optional[TreeNode]
:type x: int
:type y: int
:rtype: bool
"""
```

**JavaScript Solution:**

```
/**
* Problem: Cousins in Binary 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} root
* @param {number} x
* @param {number} y
* @return {boolean}
*/
var isCousins = function(root, x, y) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Cousins in Binary 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 isCousins(root: TreeNode | null, x: number, y: number): boolean {

};
```

**C# Solution:**

```
/*
* Problem: Cousins in Binary 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 IsCousins(TreeNode root, int x, int y) {

}
}
```

### C Solution:

```
/*
* Problem: Cousins in Binary 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 isCousins(struct TreeNode* root, int x, int y) {

}
```

### Go Solution:

```
// Problem: Cousins in Binary 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 isCousins(root *TreeNode, x int, y int) 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 isCousins(root: TreeNode?, x: Int, y: Int): 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 isCousins(_ root: TreeNode?, _ x: Int, _ y: Int) -> Bool {

}
}
```

**Rust Solution:**

```
// Problem: Cousins in Binary 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_cousins(root: Option<Rc<RefCell<TreeNode>>>, x: i32, y: i32) ->
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} root
# @param {Integer} x
# @param {Integer} y
# @return {Boolean}
def is_cousins(root, x, y)

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 $root
* @param Integer $x
* @param Integer $y
* @return Boolean
*/
function isCousins($root, $x, $y) {

}
}
```

**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 isCousins(TreeNode? root, int x, int y) {

}
}
```

**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 isCousins(root: TreeNode, x: Int, y: Int): 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_cousins(root :: TreeNode.t | nil, x :: integer, y :: integer) ::
boolean
def is_cousins(root, x, y) 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_cousins(Root :: #tree_node{} | null, X :: integer(), Y :: integer())
-> boolean().
is_cousins(Root, X, Y) ->
.
```

**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-cousins root x y)
(-> (or/c tree-node? #f) exact-integer? exact-integer? boolean?)
)
```