

# Problem 662: Maximum Width of Binary Tree

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

Given the

root

of a binary tree, return

the

maximum width

of the given tree

.

The

maximum width

of a tree is the maximum

width

among all levels.

The

width

of one level is defined as the length between the end-nodes (the leftmost and rightmost non-null nodes), where the null nodes between the end-nodes that would be present in a complete binary tree extending down to that level are also counted into the length calculation.

It is

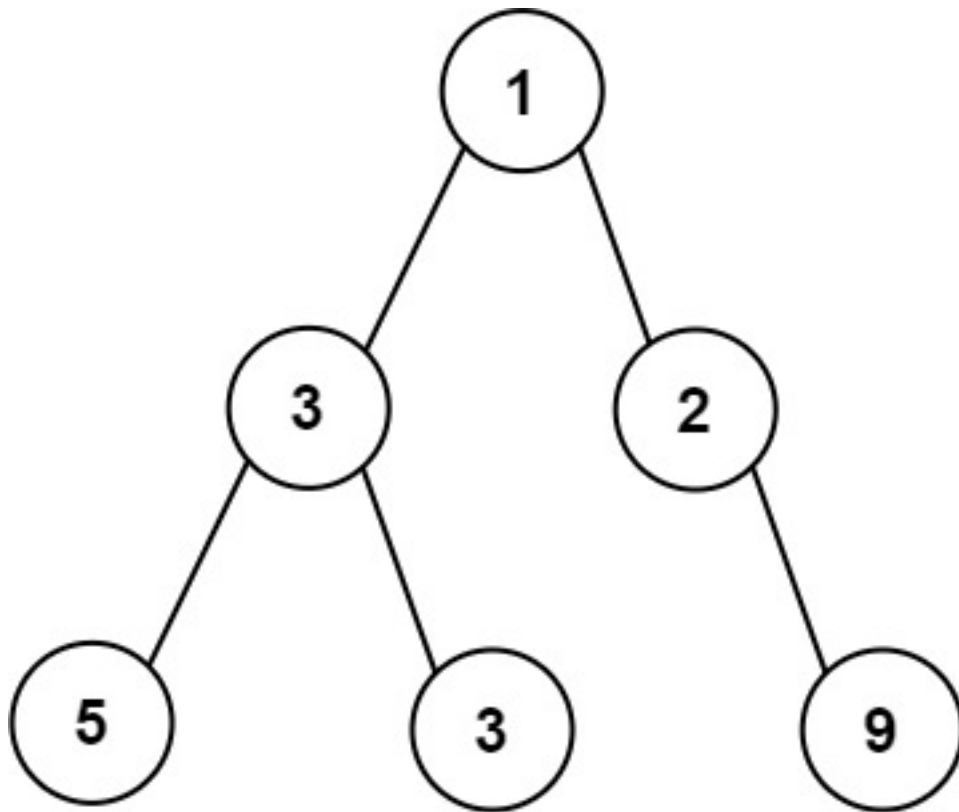
guaranteed

that the answer will in the range of a

32-bit

signed integer.

Example 1:



Input:

root = [1,3,2,5,3,null,9]

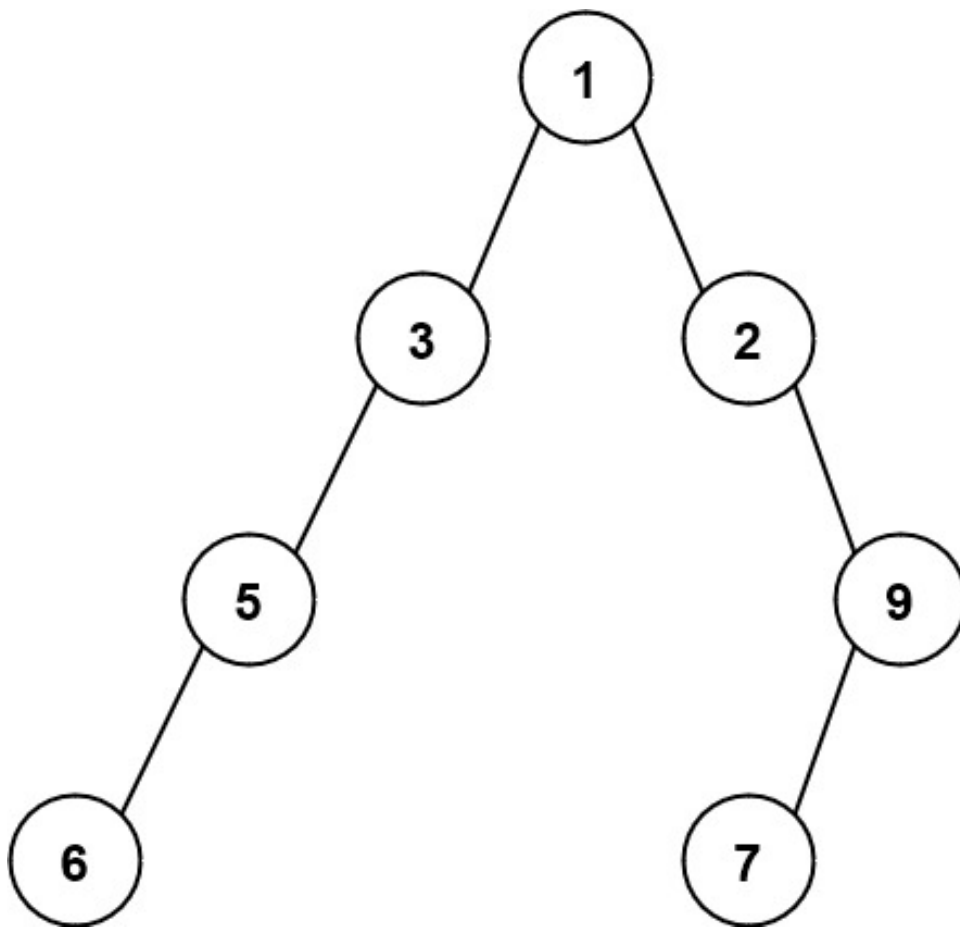
Output:

4

Explanation:

The maximum width exists in the third level with length 4 (5,3,null,9).

Example 2:



Input:

root = [1,3,2,5,null,null,9,6,null,7]

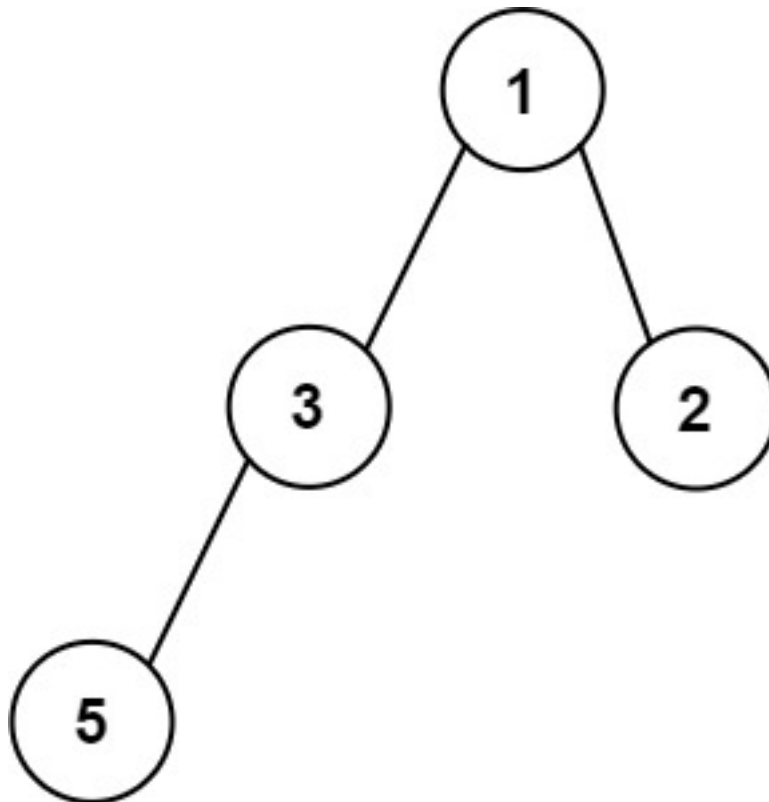
Output:

7

Explanation:

The maximum width exists in the fourth level with length 7 (6,null,null,null,null,null,7).

Example 3:



Input:

root = [1,3,2,5]

Output:

2

Explanation:

The maximum width exists in the second level with length 2 (3,2).

Constraints:

The number of nodes in the tree is in the range

[1, 3000]

.

-100 <= Node.val <= 100

## 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:
    int widthOfBinaryTree(TreeNode* root) {

    }
};
```

### 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 int widthOfBinaryTree(TreeNode root) {

}

}

```

### 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 widthOfBinaryTree(self, root: Optional[TreeNode]) -> int:

```

### 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 widthOfBinaryTree(self, root):
    """
    :type root: Optional[TreeNode]
    :rtype: int
    """

```

### 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
* @return {number}
*/
var widthOfBinaryTree = function(root) {

};

```

## 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 widthOfBinaryTree(root: TreeNode | null): number {

};

```

## 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 int WidthOfBinaryTree(TreeNode root) {

}
}

```

**C:**

```

/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
int widthOfBinaryTree(struct TreeNode* root) {

}

```

**Go:**

```

/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func widthOfBinaryTree(root *TreeNode) int {

}

```



## 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 widthOfBinaryTree(root: TreeNode?): Int {

    }
}
```

## 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 widthOfBinaryTree(_ root: TreeNode?) -> Int {

    }
}
```

## 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 width_of_binary_tree(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {

    }
}

```

## 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
# @return {Integer}
def width_of_binary_tree(root)

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 $root
 * @return Integer
 */
function widthOfBinaryTree($root) {

}

}
```

## 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 {
int widthOfBinaryTree(TreeNode? root) {

}

}
```

## 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 widthOfBinaryTree(root: TreeNode): Int = {

  }
}
```

## 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 width_of_binary_tree(root :: TreeNode.t | nil) :: integer
  def width_of_binary_tree(root) 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 width_of_binary_tree(Root :: #tree_node{} | null) -> integer().
width_of_binary_tree(Root) ->
.

```

## 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 (width-of-binary-tree root)
  (-> (or/c tree-node? #f) exact-integer?)
  )

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Maximum Width of Binary Tree
 * Difficulty: Medium
 * 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:
    int widthOfBinaryTree(TreeNode* root) {

    }

};

```

## Java Solution:

```

/**
 * Problem: Maximum Width of Binary Tree
 * Difficulty: Medium
 * 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 int widthOfBinaryTree(TreeNode root) {

    }
}

```

### Python3 Solution:

```

"""
Problem: Maximum Width of Binary Tree
Difficulty: Medium
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 widthOfBinaryTree(self, root: Optional[TreeNode]) -> int:
# 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 widthOfBinaryTree(self, root):
"""
:type root: Optional[TreeNode]
:rtype: int
"""

```

## JavaScript Solution:

```

/**
 * Problem: Maximum Width of Binary Tree
 * Difficulty: Medium
 * 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
 * @return {number}
 */
var widthOfBinaryTree = function(root) {

};

```

### TypeScript Solution:

```

/**
 * Problem: Maximum Width of Binary Tree
 * Difficulty: Medium
 * 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 widthOfBinaryTree(root: TreeNode | null): number {

};

```

### C# Solution:

```

/*
 * Problem: Maximum Width of Binary Tree
 * Difficulty: Medium
 * 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 int WidthOfBinaryTree(TreeNode root) {

}
}

```

## C Solution:

```

/*
 * Problem: Maximum Width of Binary Tree
 * Difficulty: Medium
 * 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;
* };
*/
int widthOfBinaryTree(struct TreeNode* root) {

}

```

### Go Solution:

```

// Problem: Maximum Width of Binary Tree
// Difficulty: Medium
// 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 widthOfBinaryTree(root *TreeNode) int {

}

```

### 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 widthOfBinaryTree(root: TreeNode?): Int {

}
}

```

### 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 widthOfBinaryTree(_ root: TreeNode?) -> Int {

}
}

```

### Rust Solution:

```

// Problem: Maximum Width of Binary Tree
// Difficulty: Medium
// 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 width_of_binary_tree(root: Option<Rc<RefCell<TreeNode>>>) -> i32 {

    }
}

```

## 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

```

```
# @return {Integer}
def width_of_binary_tree(root)

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 $root
 * @return Integer
 */
function widthOfBinaryTree($root) {

}

}
```

### 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 {
    int widthOfBinaryTree(TreeNode? root) {

    }
}

```

### 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 widthOfBinaryTree(root: TreeNode): Int = {

    }
}

```

### 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 width_of_binary_tree(root :: TreeNode.t | nil) :: integer
  def width_of_binary_tree(root) 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 width_of_binary_tree(Root :: #tree_node{} | null) -> integer().
width_of_binary_tree(Root) ->
.

```

### 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 (width-of-binary-tree root)
  (-> (or/c tree-node? #f) exact-integer?)
)
```