

# Problem 257: Binary Tree Paths

## Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

Given the

root

of a binary tree, return

all root-to-leaf paths in

any order

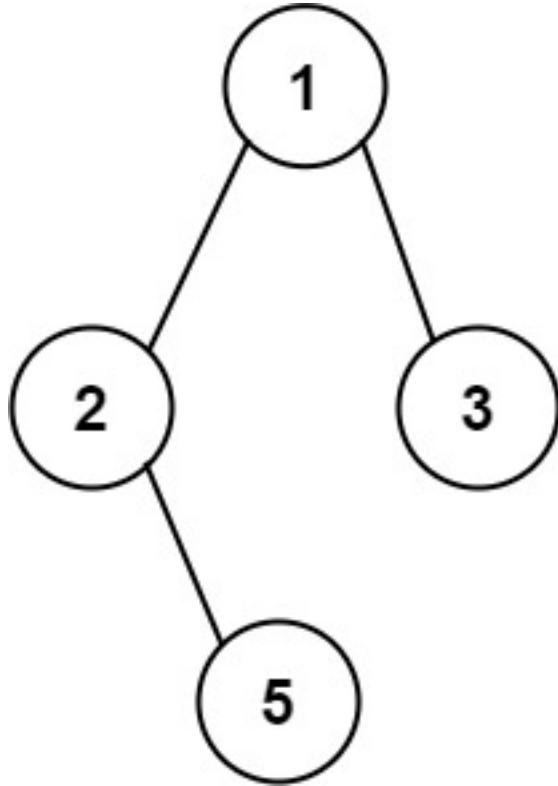
.

A

leaf

is a node with no children.

Example 1:



Input:

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

Output:

```
["1->2->5","1->3"]
```

Example 2:

Input:

```
root = [1]
```

Output:

```
["1"]
```

Constraints:

The number of nodes in the tree is in the range

[1, 100]

-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:
    vector<string> binaryTreePaths(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 List<String> binaryTreePaths(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 binaryTreePaths(self, root: Optional[TreeNode]) -> List[str]:

```

### 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 binaryTreePaths(self, root):
        """
:type root: Optional[TreeNode]
:rtype: List[str]
"""

```

### 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 {string[]}
*/
var binaryTreePaths = 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 binaryTreePaths(root: TreeNode | null): string[] {

};


```

### 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 IList<string> BinaryTreePaths(TreeNode root) {
}

}
}

```

## C:

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * struct TreeNode *left;
 * struct TreeNode *right;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
char** binaryTreePaths(struct TreeNode* root, int* returnSize) {

}

```

## Go:

```

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

```

```
}
```

## 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 binaryTreePaths(root: TreeNode?): List<String> {  
  
    }  
}
```

## 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 binaryTreePaths(_ root: TreeNode?) -> [String] {
```

```
}
```

```
}
```

## 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 binary_tree_paths(root: Option<Rc<RefCell<TreeNode>>>) -> Vec<String> {
        Vec::new()
    }
}
```

## 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 {String[]}
def binary_tree_paths(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 $root
 * @return String[]
 */
function binaryTreePaths($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 {
List<String> binaryTreePaths(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 binaryTreePaths(root: TreeNode): List[String] = {

}
}

```

### 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 binary_tree_paths(TreeNode.t() | nil) :: [String.t]
def binary_tree_paths(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 binary_tree_paths(Root :: #tree_node{} | null) ->  
[unicode:unicode_binary()].  
binary_tree_paths(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 (binary-tree-paths root)  
(-> (or/c tree-node? #f) (listof string?))  
)
```

## Solutions

### C++ Solution:

```

/*
 * Problem: Binary Tree Paths
 * Difficulty: Easy
 * Tags: string, tree, search
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * 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) {}
 *     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:
    vector<string> binaryTreePaths(TreeNode* root) {
        }
    };
}

```

## Java Solution:

```

/**
 * Problem: Binary Tree Paths
 * Difficulty: Easy
 * Tags: string, tree, search
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * 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() {}
* 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 List<String> binaryTreePaths(TreeNode root) {

}
}

```

### Python3 Solution:

```

"""
Problem: Binary Tree Paths
Difficulty: Easy
Tags: string, tree, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
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 binaryTreePaths(self, root: Optional[TreeNode]) -> List[str]:
        # 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 binaryTreePaths(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: List[str]
        """

```

## JavaScript Solution:

```
/**
 * Problem: Binary Tree Paths
 * Difficulty: Easy
 * Tags: string, tree, search
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * 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 {string[]}
 */
```

```
var binaryTreePaths = function(root) {  
};
```

### TypeScript Solution:

```
/**  
 * Problem: Binary Tree Paths  
 * Difficulty: Easy  
 * Tags: string, tree, search  
 *  
 * Approach: String manipulation with hash map or two pointers  
 * Time Complexity: O(n) or O(n log n)  
 * 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 binaryTreePaths(root: TreeNode | null): string[] {  
};
```

### C# Solution:

```
/*  
 * Problem: Binary Tree Paths  
 * Difficulty: Easy  
 * Tags: string, tree, search
```

```

/*
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * 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 IList<string> BinaryTreePaths(TreeNode root) {
 *
 *     }
 * }
 */

```

## C Solution:

```

/*
 * Problem: Binary Tree Paths
 * Difficulty: Easy
 * Tags: string, tree, search
 *
 * Approach: String manipulation with hash map or two pointers
 * Time Complexity: O(n) or O(n log n)
 * 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;
* };
*/
/***
* Note: The returned array must be malloced, assume caller calls free().
*/
char** binaryTreePaths(struct TreeNode* root, int* returnSize) {

}

```

### Go Solution:

```

// Problem: Binary Tree Paths
// Difficulty: Easy
// Tags: string, tree, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// 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 binaryTreePaths(root *TreeNode) []string {

}

```

### 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 binaryTreePaths(root: TreeNode?): List<String> {
}
}

```

### 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 binaryTreePaths(_ root: TreeNode?) -> [String] {
}
}

```

### Rust Solution:

```

// Problem: Binary Tree Paths
// Difficulty: Easy
// Tags: string, tree, search
//
// Approach: String manipulation with hash map or two pointers

```

```

// Time Complexity: O(n) or O(n log n)
// 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 binary_tree_paths(root: Option<Rc<RefCell<TreeNode>>>) -> Vec<String> {
        Vec::new()
    }
}

```

## 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 {String[]}
def binary_tree_paths(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 $root
 * @return String[]
 */
function binaryTreePaths($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 {
List<String> binaryTreePaths(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 binaryTreePaths(root: TreeNode): List[String] = {

}
}

```

### 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 binary_tree_paths(TreeNode.t() | nil) :: [String.t]
def binary_tree_paths(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 binary_tree_paths(Root :: #tree_node{} | null) ->  
[unicode:unicode_binary()].  
binary_tree_paths(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 (binary-tree-paths root)  
(-> (or/c tree-node? #f) (listof string?)))  
)
```