

Problem 2096: Step-By-Step Directions From a Binary Tree Node to Another

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given the

root

of a

binary tree

with

n

nodes. Each node is uniquely assigned a value from

1

to

n

. You are also given an integer

startValue

representing the value of the start node

s

, and a different integer

destValue

representing the value of the destination node

t

.

Find the

shortest path

starting from node

s

and ending at node

t

. Generate step-by-step directions of such path as a string consisting of only the

uppercase

letters

'L'

,

'R'

, and

'U'

. Each letter indicates a specific direction:

'L'

means to go from a node to its

left child

node.

'R'

means to go from a node to its

right child

node.

'U'

means to go from a node to its

parent

node.

Return

the step-by-step directions of the

shortest path

from node

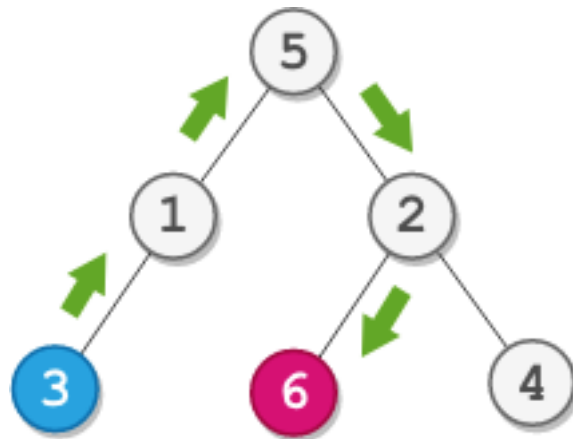
s

to node

t

.

Example 1:



Input:

root = [5,1,2,3,null,6,4], startValue = 3, destValue = 6

Output:

"UURL"

Explanation:

The shortest path is: 3 → 1 → 5 → 2 → 6.

Example 2:



Input:

root = [2,1], startValue = 2, destValue = 1

Output:

"L"

Explanation:

The shortest path is: 2 → 1.

Constraints:

The number of nodes in the tree is

n

.

$2 \leq n \leq 10$

5

$1 \leq \text{Node.val} \leq n$

All the values in the tree are

unique

.

$1 \leq \text{startValue}, \text{destValue} \leq n$

$\text{startValue} \neq \text{destValue}$

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:
    string getDirections(TreeNode* root, int startValue, int destValue) {

    }
};

```

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 String getDirections(TreeNode root, int startValue, int destValue) {

    }
}

```

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 getDirections(self, root: Optional[TreeNode], startValue: int, destValue:
int) -> 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 getDirections(self, root, startValue, destValue):
"""
:type root: Optional[TreeNode]
:type startValue: int
:type destValue: int
:rtype: 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
 * @param {number} startValue
 * @param {number} destValue
```

```

* @return {string}
*/
var getDirections = function(root, startValue, destValue) {

};

```

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 getDirections(root: TreeNode | null, startValue: number, destValue:
number): 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 string GetDirections(TreeNode root, int startValue, int destValue) {

}

}

```

C:

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
char* getDirections(struct TreeNode* root, int startValue, int destValue) {

}

```

Go:

```

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func getDirections(root *TreeNode, startValue int, destValue int) 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 getDirections(root: TreeNode?, startValue: Int, destValue: Int): 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 getDirections(_ root: TreeNode?, _ startValue: Int, _ destValue: Int) ->
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 get_directions(root: Option<Rc<RefCell<TreeNode>>>, start_value: i32,
dest_value: i32) -> String {

}
}

```

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} start_value
# @param {Integer} dest_value
# @return {String}

def get_directions(root, start_value, dest_value)

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 $startValue
 * @param Integer $destValue
 * @return String
 */
function getDirections($root, $startValue, $destValue) {

}

}
```

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 {
String getDirections(TreeNode? root, int startValue, int destValue) {
```

```
}  
}
```

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 getDirections(root: TreeNode, startValue: Int, destValue: Int): 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 get_directions(root :: TreeNode.t | nil, start_value :: integer,  
    dest_value :: integer) :: String.t  
  def get_directions(root, start_value, dest_value) 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 get_directions(Root :: #tree_node{} | null, StartValue :: integer(),
DestValue :: integer()) -> unicode:unicode_binary().
get_directions(Root, StartValue, DestValue) ->
.

```

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 (get-directions root startValue destValue)
(-> (or/c tree-node? #f) exact-integer? exact-integer? string?)
)

```

Solutions

C++ Solution:

```
/*
 * Problem: Step-By-Step Directions From a Binary Tree Node to Another
 * Difficulty: Medium

```

```

* Tags: string, tree, graph, 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) {
// 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:
string getDirections(TreeNode* root, int startValue, int destValue) {

}
};

```

Java Solution:

```

/**
* Problem: Step-By-Step Directions From a Binary Tree Node to Another
* Difficulty: Medium
* Tags: string, tree, graph, 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() {
// 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 String getDirections(TreeNode root, int startValue, int destValue) {

}

}

```

Python3 Solution:

```

"""
Problem: Step-By-Step Directions From a Binary Tree Node to Another
Difficulty: Medium
Tags: string, tree, graph, 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 getDirections(self, root: Optional[TreeNode], startValue: int, destValue:
int) -> 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 getDirections(self, root, startValue, destValue):
"""
:type root: Optional[TreeNode]
:type startValue: int
:type destValue: int
:rtype: str
"""

```

JavaScript Solution:

```

/**
 * Problem: Step-By-Step Directions From a Binary Tree Node to Another
 * Difficulty: Medium
 * Tags: string, tree, graph, 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
 * @param {number} startValue
 * @param {number} destValue
 * @return {string}
 */
var getDirections = function(root, startValue, destValue) {

};

```

TypeScript Solution:

```

/**
 * Problem: Step-By-Step Directions From a Binary Tree Node to Another
 * Difficulty: Medium
 * Tags: string, tree, graph, 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 getDirections(root: TreeNode | null, startValue: number, destValue:
number): string {

}
};

```

C# Solution:

```

/*
 * Problem: Step-By-Step Directions From a Binary Tree Node to Another
 * Difficulty: Medium
 * Tags: string, tree, graph, 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 string GetDirections(TreeNode root, int startValue, int destValue) {

    }
}

```

C Solution:

```
/*
 * Problem: Step-By-Step Directions From a Binary Tree Node to Another
 * Difficulty: Medium
 * Tags: string, tree, graph, 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;
 * };
 */
char* getDirections(struct TreeNode* root, int startValue, int destValue) {

}
```

Go Solution:

```
// Problem: Step-By-Step Directions From a Binary Tree Node to Another
// Difficulty: Medium
// Tags: string, tree, graph, 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 getDirections(root *TreeNode, startValue int, destValue int) 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 getDirections(root: TreeNode?, startValue: Int, destValue: Int): 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 getDirections(_ root: TreeNode?, _ startValue: Int, _ destValue: Int) ->
```

```
String {

}

}
```

Rust Solution:

```
// Problem: Step-By-Step Directions From a Binary Tree Node to Another
// Difficulty: Medium
// Tags: string, tree, graph, 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 get_directions(root: Option<Rc<RefCell<TreeNode>>>, start_value: i32,
        dest_value: i32) -> String {

    }

}
```

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} start_value
# @param {Integer} dest_value
# @return {String}
def get_directions(root, start_value, dest_value)

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 $startValue
 * @param Integer $destValue
 * @return String
 */
function getDirections($root, $startValue, $destValue) {
```

```
}  
}
```

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 {  
  String getDirections(TreeNode? root, int startValue, int destValue) {  
  
  }  
}
```

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 getDirections(root: TreeNode, startValue: Int, destValue: Int): 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 get_directions(root :: TreeNode.t() | nil, start_value :: integer,
    dest_value :: integer) :: String.t
  def get_directions(root, start_value, dest_value) 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 get_directions(Root :: #tree_node{} | null, StartValue :: integer(),
  DestValue :: integer()) -> unicode:unicode_binary().
get_directions(Root, StartValue, DestValue) ->
.

```

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 (get-directions root startValue destValue)
  (-> (or/c tree-node? #f) exact-integer? exact-integer? string?)
  )
```