

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

Leetcode Link

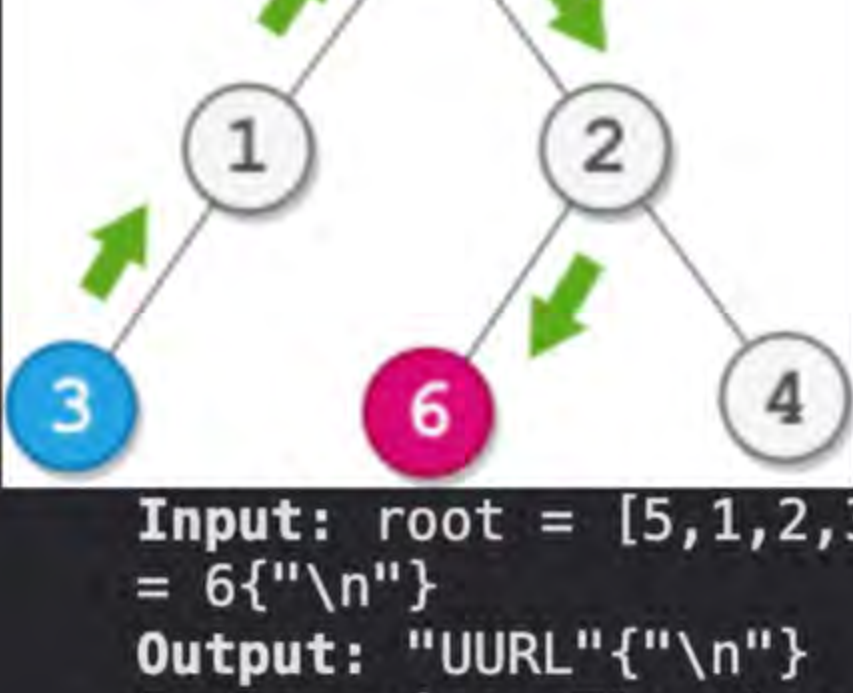
You are given the **root** of a **binary tree** with n nodes. Each node is uniquely assigned a value from 1 to n . You are also given an integer s representing the value of the start node, and a different integer $destValue$ representing the value of the destination node t .

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** $\{ 'L', 'R', \text{ and } 'U' \}$. 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:

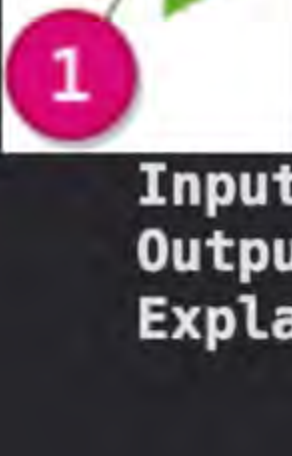


Input: root = [5,1,2,3,null,6,4], startValue = 3, destValue = 6

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:

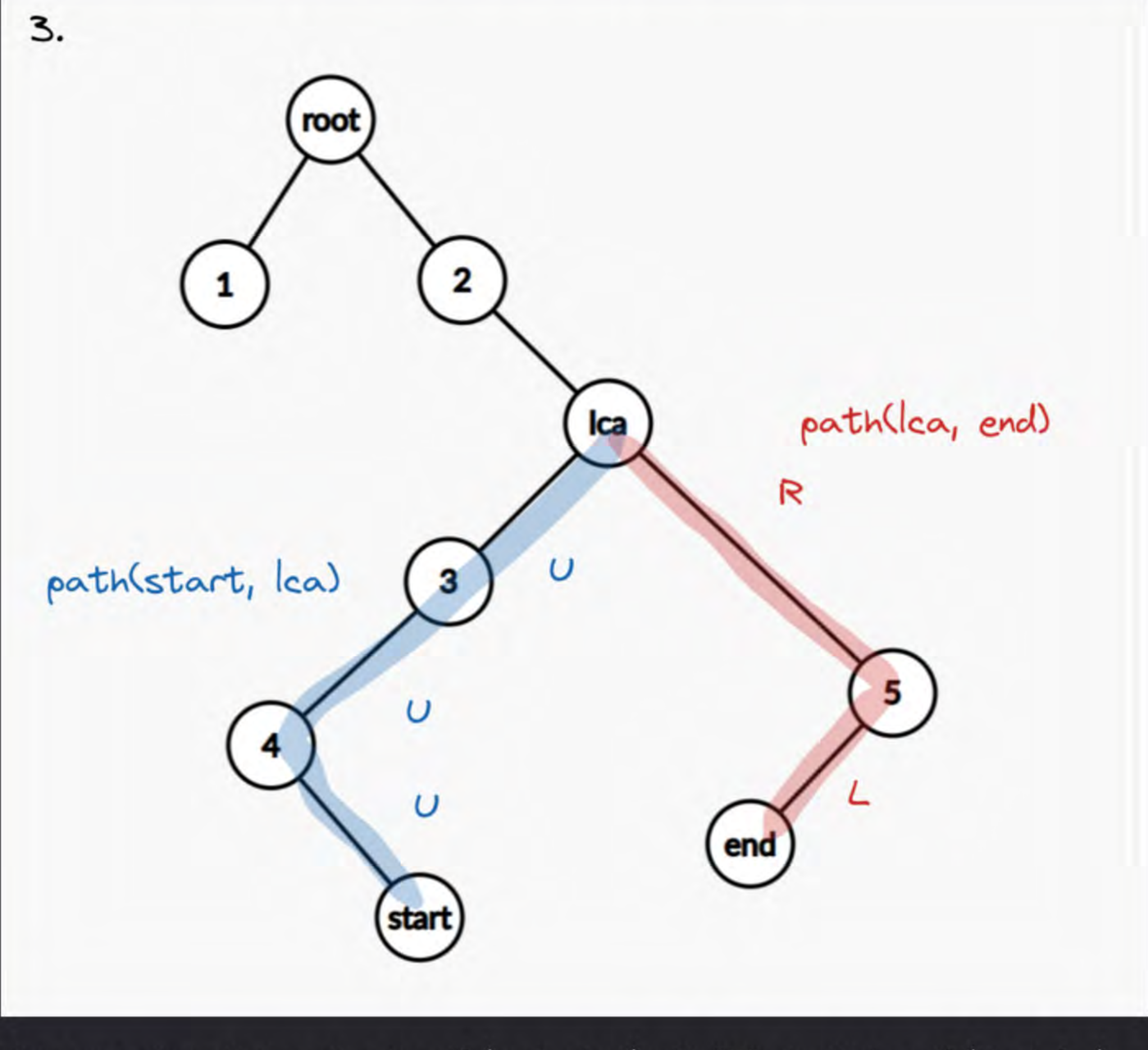
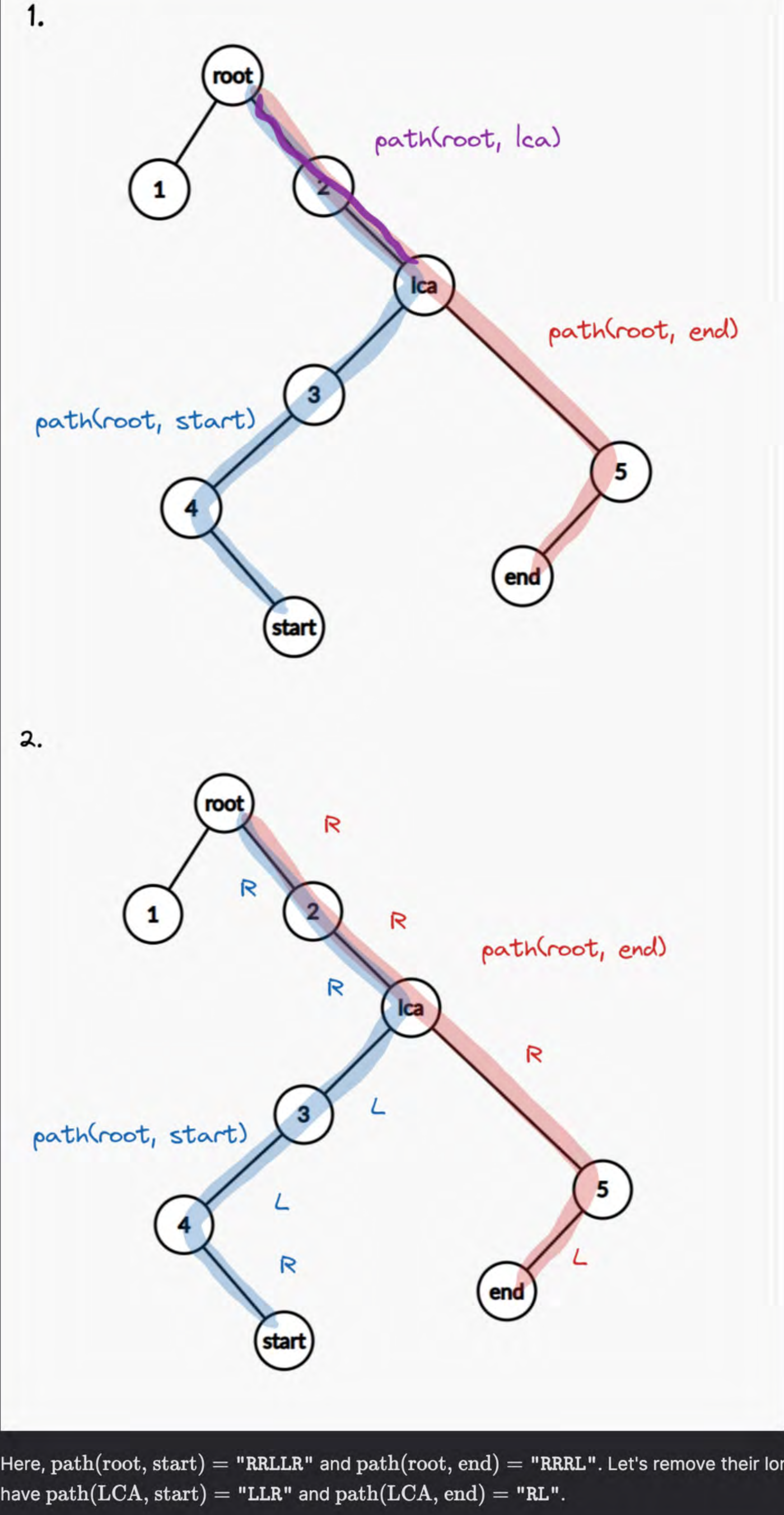
- The number of nodes in the tree is n .
- $2 \leq n \leq 10^5$
- $1 \leq \text{Node.val} \leq n$
- All the values in the tree are **unique**.
- $1 \leq \text{startValue}, \text{destValue} \leq n$
- $\text{startValue} \neq \text{destValue}$

Solution

Let $\text{path}(\text{node1}, \text{node2})$ denote the path from node1 to node2 .

First consider the case where $\text{path}(\text{start}, \text{end})$ goes through the root. Let's split this into $\text{path}(\text{start}, \text{root}) + \text{path}(\text{root}, \text{end})$. We can perform a DFS (**depth first search**) to get $\text{path}(\text{root}, \text{end})$. This path consists of 'L's and 'R's. We can do another DFS to get $\text{path}(\text{root}, \text{start})$. Replacing the 'L's and 'R's of $\text{path}(\text{root}, \text{start})$ with 'U's gives us $\text{path}(\text{start}, \text{root})$. Now we can concatenate $\text{path}(\text{start}, \text{root})$ and $\text{path}(\text{root}, \text{end})$ to get the answer.

In the general case, $\text{path}(\text{start}, \text{end})$ may not go through the root. Notice that this path goes up a non-negative number of times ('U's) before going down a non-negative number of times ('L's or 'R's). The highest node in this path is known as the **LCA (lowest common ancestor)** of start and end.



Then we replace all characters in $\text{path}(\text{root}, \text{start})$ with 'U's to obtain $\text{path}(\text{start}, \text{lca}) = \text{"UUU"}$. Finally, we get $\text{path}(\text{start}, \text{end}) = \text{path}(\text{start}, \text{LCA}) + \text{path}(\text{LCA}, \text{end}) = \text{"UUU"} + \text{"RL"} = \text{"UUURL"}$.

Time complexity

Each DFS takes $\mathcal{O}(n)$ and our string operations never happen on strings exceeding length $\mathcal{O}(n)$. The time complexity is $\mathcal{O}(n)$.

Space complexity

The strings never exceed length $\mathcal{O}(n)$. The space complexity is $\mathcal{O}(n)$.

C++ Solution

```
1 /**
2  * Definition for a binary tree node.
3  * struct TreeNode {
4  *     int val;
5  *     TreeNode *left;
6  *     TreeNode *right;
7  *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8  *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9  *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
10 * right(right) {}
11 * };
12 */
13 class Solution {
14 public:
15     void getPath(TreeNode *cur, int targetValue, string &path, string &ans) {
16         if (!cur)
17             return;
18         if (cur->val == targetValue)
19             ans = path;
20         path.push_back('L');
21         getPath(cur->left, targetValue, path, ans);
22         path.back() = 'R';
23         getPath(cur->right, targetValue, path, ans);
24         path.pop_back();
25     }
26
27     string getDirections(TreeNode *root, int startValue, int destValue) {
28         string tmpPath, startPath, destPath;
29         getPath(root, startValue, tmpPath, startPath);
30         getPath(root, destValue, tmpPath, destPath);
31         // Find the first point at which the paths diverge
32         auto [itr1, itr2] = mismatch(startPath.begin(), startPath.end(),
33                                     destPath.begin(), destPath.end());
34         return string(startPath.end() - itr1, 'U') +
35                string(itr2, destPath.end());
36     }
37 };
```

Java Solution

```
1 /**
2  * Definition for a binary tree node.
3  * public class TreeNode {
4  *     int val;
5  *     TreeNode left;
6  *     TreeNode right;
7  *     TreeNode() {}
8  *     TreeNode(int val) { this.val = val; }
9  *     TreeNode(int val, TreeNode left, TreeNode right) {
10 *         this.val = val;
11 *         this.left = left;
12 *         this.right = right;
13 *     }
14 * }
15 */
16 class Solution {
17     // We use StringBuilder instead of String because String is immutable,
18     // so appending a character takes O(length of string).
19     // ans is a String[] instead of String because in Java, arrays are passed by reference.
20     void getPath(TreeNode cur, int targetValue, StringBuilder path, String[] ans) {
21         if (cur == null)
22             return;
23         if (cur.val == targetValue)
24             ans[0] = path.toString();
25         int strLen = path.length();
26         path.append("L");
27         getPath(cur.left, targetValue, path, ans);
28         path.replace(strLen, strLen+1, "R");
29         getPath(cur.right, targetValue, path, ans);
30         path.delete(strLen, strLen+1);
31     }
32
33     public String getDirections(TreeNode root, int startValue, int destValue) {
34         StringBuilder tmpPath = new StringBuilder();
35         String[] startPath = {"", ""}, destPath = {"", ""};
36         // Find the first point at which the paths diverge
37         getPath(root, startValue, tmpPath, startPath);
38         getPath(root, destValue, tmpPath, destPath);
39         int i = 0;
40         while (i < Math.min(startPath[0].length(), destPath[0].length()) && startPath[0].charAt(i) == destPath[0].charAt(i))
41             i++;
42         return "U".repeat(startPath[0].length()-i) + destPath[0].substring(i);
43     }
44 }
```

Python Solution

```
1 # Definition for a binary tree node.
2 # class TreeNode:
3 #     def __init__(self, val=0, left=None, right=None):
4 #         self.val = val
5 #         self.left = left
6 #         self.right = right
7 class Solution:
8     def getDirections(self, root: Optional[TreeNode], startValue: int, destValue: int) -> str:
9         # ans is a list so that it passes by reference
10         def getPath(cur, targetValue, path, ans):
11             if cur is None:
12                 return
13             if cur.val == targetValue:
14                 ans.append(''.join(path))
15                 path.append('L')
16                 getPath(cur.left, targetValue, path, ans)
17                 path[-1] = 'R'
18                 getPath(cur.right, targetValue, path, ans)
19                 path.pop(-1)
20
21         tmpPath = []
22         startPath = []
23         destPath = []
24         getPath(root, startValue, tmpPath, startPath)
25         getPath(root, destValue, tmpPath, destPath)
26         startPath = startPath[0]
27         destPath = destPath[0]
28
29         # Find the first point at which the paths diverge
30         i = 0
31         while i < min(len(startPath), len(destPath)) and startPath[i] == destPath[i]:
32             i += 1
33         return 'U'*(len(startPath)-i) + destPath[i:]
```

Got a question? Ask the Teaching Assistant anything you don't understand.