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

Leetcode Link

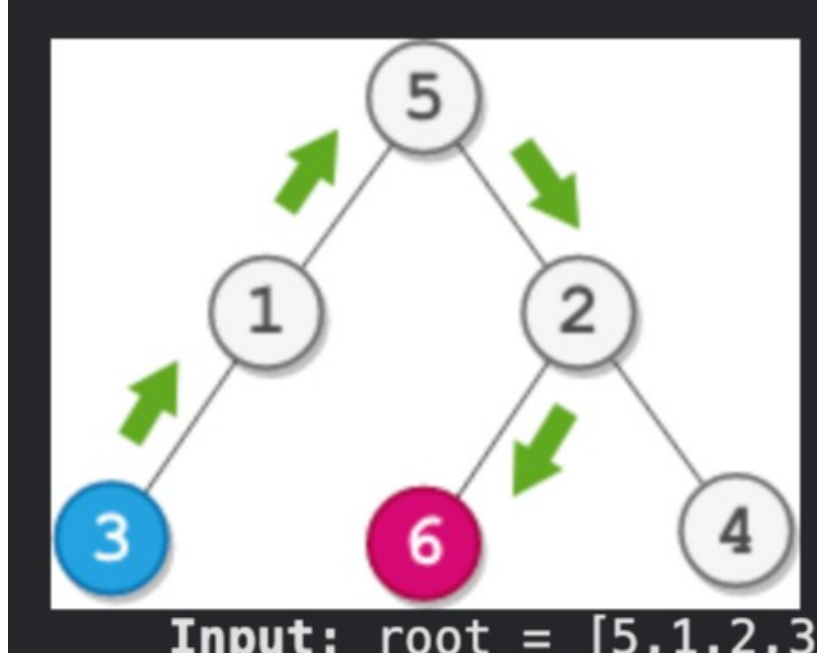
You are given the `root` of a **binary tree** with{" "} `n` nodes. Each node is uniquely assigned a value from{" "} `1` to `n`. You are also given an integer{" "} `startValue` representing the value of the start node{" "} `s`, and a different integer `destValue` representing the value of the destination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

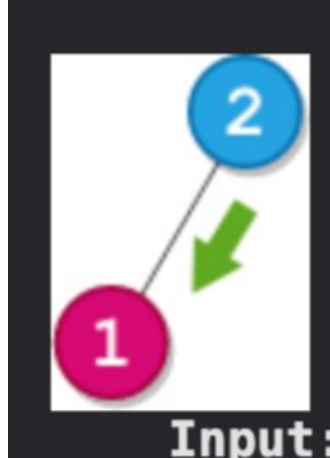
Return{" "} *the step-by-step directions of the **shortest path** from node{" "}* `s` *to node* `t`.

Example 1:



```
Input: root = [5,1,2,3,null,6,4], startValue = 3, destValue
= 6{"\n"}
Output: "UURL"{"\n"}
Explanation: The shortest path is: 3 → 1 → 5 → 2 → 6.{"\n"}
```

Example 2:



```
Input: root = [2,1], startValue = 2, destValue = 1{"\n"}
Output: "L"{"\n"}
Explanation: The shortest path is: 2 → 1.{"\n"}
```

Constraints:

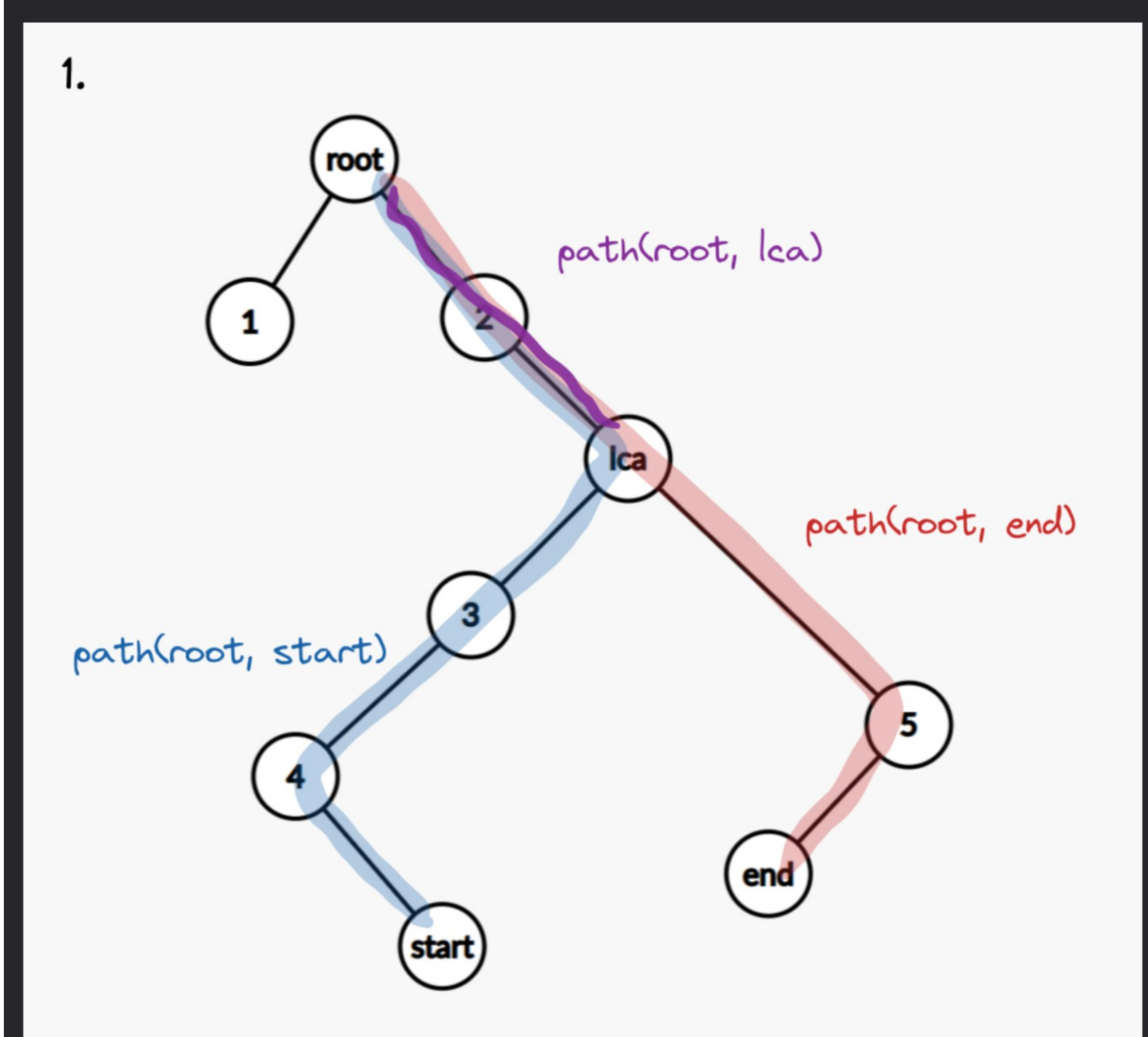
- The number of nodes in the tree is `n`.
- `2 <= n <= 10`^{5}
- `1 <= Node.val <= n`
- All the values in the tree are **unique**.
- `1 <= startValue, destValue <= n`
- `startValue != destValue`

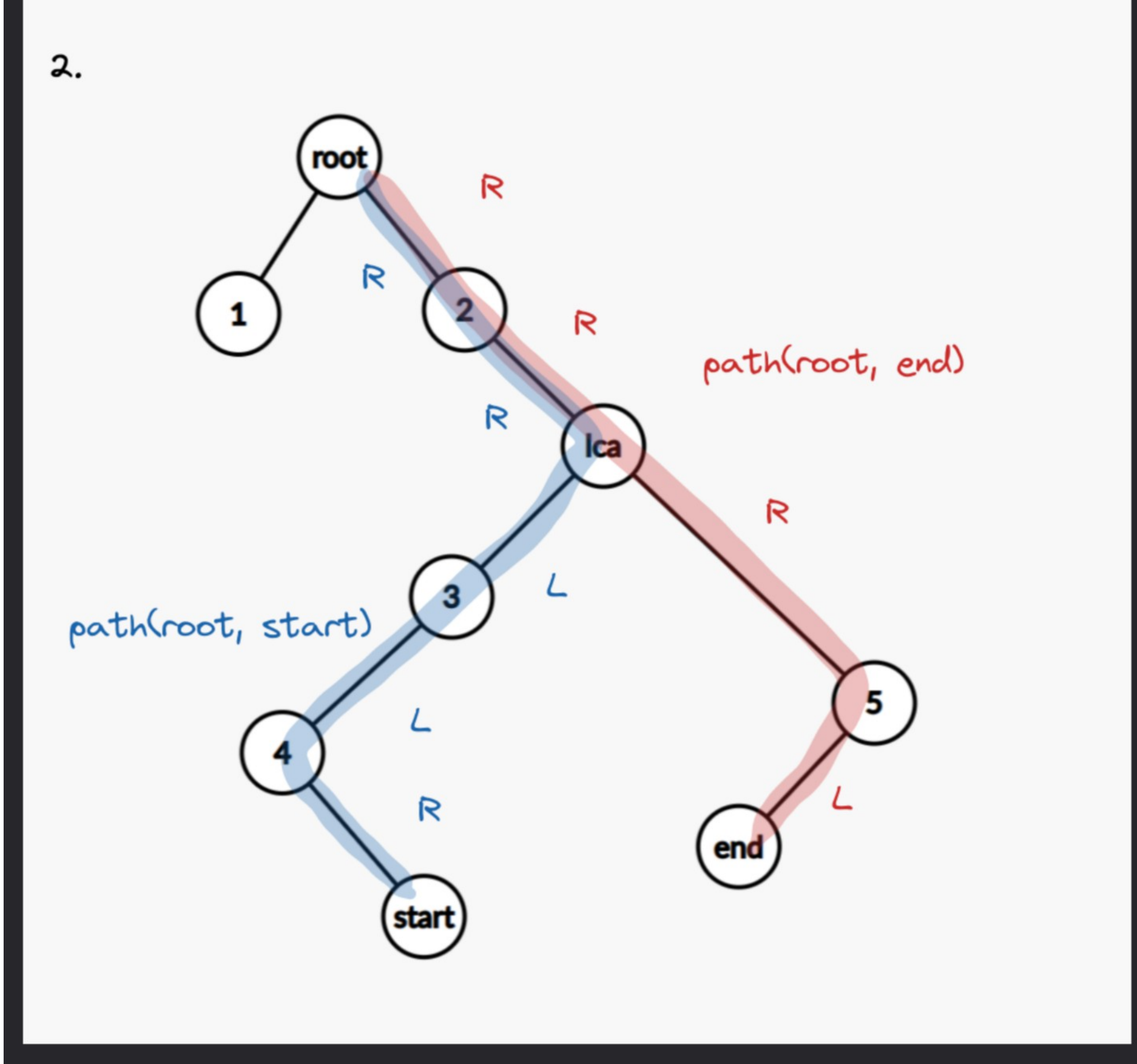
Solution

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

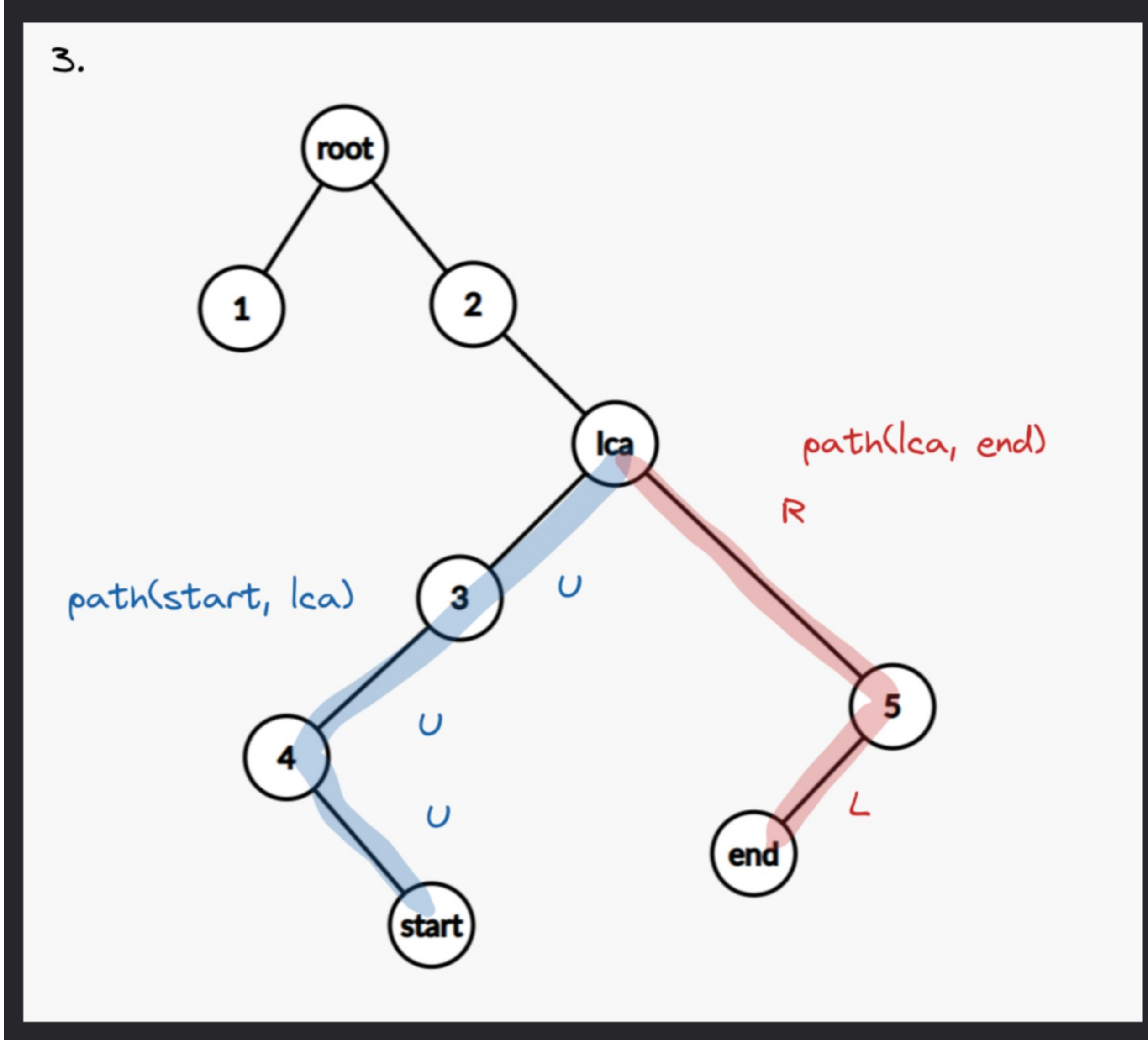
First consider the case where $\mathrm{path}(\mathrm{start}, \mathrm{end})$ goes through the root. Let's split this into $\mathrm{path}(\mathrm{start}, \mathrm{root}) + \mathrm{path}(\mathrm{root}, \mathrm{end})$. We can perform a DFS (depth first search) to get $\mathrm{path}(\mathrm{root}, \mathrm{end})$.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

In the general case, $\mathrm{path}(\mathrm{start}, \mathrm{end})$ 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.





Here, $\mathrm{path}(\mathrm{root}, \mathrm{start}) = \texttt{"RRLLR"}$ and $\mathrm{path}(\mathrm{root}, \mathrm{end}) = \texttt{"RRRL"}$. Let's remove their longest common prefix, which is `"RR"`. We have $\mathrm{path}(\mathrm{LCA}, \mathrm{start}) = \texttt{"LLR"}$ and $\mathrm{path}(\mathrm{LCA}, \mathrm{end}) = \texttt{"RL"}$.



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

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

```cpp
/**
 * 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:
    void getPath(TreeNode *cur, int targetValue, string &path, string &ans) {
        if (!cur)
            return;
        if (cur->val == targetValue)
            ans = path;
        path.push_back('L');
        getPath(cur->left, targetValue, path, ans);
        path.back() = 'R';
        getPath(cur->right, targetValue, path, ans);
        path.pop_back();
    }

    string getDirections(TreeNode *root, int startValue, int destValue) {
        string tmpPath, startPath, destPath;
        getPath(root, startValue, tmpPath, startPath);
        getPath(root, destValue, tmpPath, destPath);
        // Find the first point at which the paths diverge
        auto [itr1, itr2] = mismatch(startPath.begin(), startPath.end(),
                                     destPath.begin(), destPath.end());
        return string(startPath.end() - itr1, 'U') +
               string(itr2, destPath.end());
    }
};
```

Java Solution

```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 {
    // We use StringBuilder instead of String because String is immutable,
    // so appending a character takes O(length of string).
    // ans is a String[1] instead of String because in Java, arrays are passed by reference.
    void getPath(TreeNode cur, int targetValue, StringBuilder path, String[] ans) {
        if (cur == null)
            return;
        if (cur.val == targetValue)
            ans[0] = path.toString();
        int strLen = path.length();
        path.append("L");
        getPath(cur.left, targetValue, path, ans);
        path.replace(strLen, strLen+1, "R");
        getPath(cur.right, targetValue, path, ans);
        path.delete(strLen, strLen+1);
    }
    public String getDirections(TreeNode root, int startValue, int destValue) {
        StringBuilder tmpPath = new StringBuilder();
        String[] startPath = {""}, destPath = {" "};
        // Find the first point at which the paths diverge
        getPath(root, startValue, tmpPath, startPath);
        getPath(root, destValue, tmpPath, destPath);
        int i = 0;
        while (i < Math.min(startPath[0].length(), destPath[0].length()) && startPath[0].charAt(i) == destPath[0].charAt(i))
            i++;
        return "U".repeat(startPath[0].length()-i) + destPath[0].substring(i);
    }
}
```

Python Solution

```python
# 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:
        # ans is a list so that it passes by reference
        def getPath(cur, targetValue, path, ans):
            if cur is None:
                return
            if cur.val == targetValue:
                ans.append(''.join(path));
            path.append('L');
            getPath(cur.left, targetValue, path, ans)
            path[-1] = 'R'
            getPath(cur.right, targetValue, path, ans)
            path.pop(-1)

        tmpPath = []
        startPath = []
        destPath = []
        getPath(root, startValue, tmpPath, startPath)
        getPath(root, destValue, tmpPath, destPath)
        startPath = startPath[0]
        destPath = destPath[0]

        # Find the first point at which the paths diverge
        i = 0
        while i < min(len(startPath), len(destPath)) and startPath[i] == destPath[i]:
            i += 1
        return 'U'*(len(startPath)-i) + destPath[i:]
```

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