5944. Step-By-Step Directions From a Binary Tree Node to Another

My Submissions (/contest/weekly-contest-270/problems/step-by-step-directions-from-a-binary-tree-node-to-another/submissions/)

Back to Contest (/contest/weekly-contest-270/)

User Accepted:	0
User Tried:	0
Total Accepted:	0
Total Submissions:	0
Difficulty:	Medium

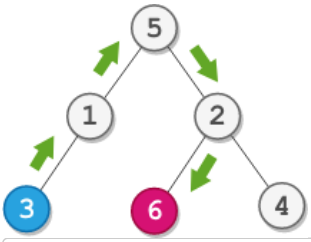
You are given the `root` of a **binary tree** with `n` nodes. Each node is uniquely assigned a value from `1` to `n`. You are also given an integer `startValue` representing the value of the start node `s`, and a different integer `destValue` representing the value of the destination node `t`.

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.

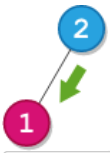
Return *the step-by-step directions of the **shortest path** from node `s` to node `t`.*

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 <= n <= 10^5`
- `1 <= Node.val <= n`
- All the values in the tree are **unique**.
- `1 <= startValue, destValue <= n`
- `startValue != destValue`

JavaScript

```
1  const aeq = (a, b) => a.length === b.length && a.every((v, i) => v[0] === b[i][0]);
2
3  let start, dest;
4  const getDirections = (root, startValue, destValue) => {
```

```
    start = startValue, dest = destValue;
    let [ps, pd] = getAllPathNew(root);
    ps.sort((x, y) => x.length - y.length);
    pd.sort((x, y) => x.length - y.length);
    let as = ps[0], ad = pd[0];
    let startI = -1, destI = -1;
    for (let i = 0; i < as.length; i++) {
        if (as[i][0] == start) {
            startI = i;
            break;
        }
    }
    for (let i = 0; i < ad.length; i++) {
        if (ad[i][0] == dest) {
            destI = i;
            break;
        }
    }
    if (aeq(as, ad)) {
        if (startI <= destI) {
            return ad.slice(startI + 1, destI + 1).map(x => x[1]).join("");
        } else {
            return 'U'.repeat(startI - destI);
        }
    }
    let lca = -1;
    for (let i = startI; i >= 0; i--) {
        if (i < ad.length && as[i][0] == ad[i][0]) {
            lca = i;
            break;
        }
    }
    let left = 'U'.repeat(lca == -1 ? as.length - 1 : startI - lca), right = '';
    let rstart = lca == -1 ? 0 : lca + 1;
    if (rstart <= destI) {
        for (let i = rstart; i <= destI; i++) {
            if (!ad[i][1]) continue;
            right += ad[i][1];
        }
    } else {
        right = "U".repeat(rstart - destI - 1);
    }
    return left + right;
};

const getAllPathNew = (root) => {
    let resS = [], resD = [];
    let path = [];
    dfs(root, path, resS, resD);
    return [resS, resD];
};

const dfs = (node, path, resS, resD, dir) => {
    if (!node) return;
    path.push([node.val, dir]);
    if (!node.left && !node.right) {
        let tmp = [...path];
        let se = new Set(tmp.map(x => x[0]));
        if (se.has(start)) resS.push(tmp);
        if (se.has(dest)) resD.push(tmp);
    }
    dfs(node.left, path, resS, resD, 'L');
    dfs(node.right, path, resS, resD, 'R');
    path.pop();
};

const pr = console.log;

```

☐ **Custom Testcase** Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/597196273/) More Details > (/submissions/detail/597196273/)

Share your acceptance!