```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public boolean hasPathSum(TreeNode node, int sum) {
        if (node == null)
            return false;
        if (node.left == null && node.right == null) {
            return node.val == sum;
        }
        return hasPathSum(node.left, sum - node.val) || hasPathSum(node.right, sum - node.val);
    }
}
```

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22`,

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \      \
7    2      1
```

return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22`,

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \    / \
7    2  5   1
```

Return:

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public List<List<Integer>> pathSum(TreeNode root, int sum) {
        List<List<Integer>> res = new ArrayList<>();
        if (root == null)
            return res;
        helper(res, new ArrayList<>(), root, sum);
        return res;
    }

    private void helper(List<List<Integer>> res, List<Integer> tmp, TreeNode node, int sum) {
        if (node == null) {
            return;
        }

        if (node.left == null && node.right == null && node.val == sum) {
            tmp.add(node.val);
            res.add(new ArrayList<>(tmp));
            tmp.remove(tmp.size()-1);
            return;
        }

        tmp.add(node.val);
        helper(res, tmp, node.left, sum - node.val);
        helper(res, tmp, node.right, sum - node.val);
        tmp.remove(tmp.size()-1);
    }
}
```