997. Find the Town Judge

We will add every trust relationship to memo. Then we will find the candidate that no one trusts him. Then we will check this candidate is valid. TC is O(n)

```
from collections import defaultdict
class Solution:
    def findJudge(self, N: int, trust: List[List[int]]) -> int:
        memo = defaultdict(set)
        for f, t in trust:
            memo[f].add(t)

        candidate = 1
        for i in range(2, N + 1):
            if i in memo[candidate]:
                candidate = i

        for i in range(1, N + 1):
            if i != candidate and candidate not in memo[i]:
                return -1
        return candidate if len(memo[candidate]) == 0 else -1
```

814. Binary Tree Pruning

We will traverse the tree in post order if left branch tree is None and right branch tree is None and cur is zero, we will return 0. Else we will return 1. TC is O(n)

```
class Solution:
    def pruneTree(self, root: TreeNode) -> TreeNode:
        def traverse(node):
            if not node:
                return 0
            left = traverse(node.left)
            right = traverse(node.right)
            if left == 0 and right == 0 and node.val == 0:
                return 0
            else:
                if left == 0:
                    node.left = None
                if right == 0:
                    node.right = None
                return 1

        if traverse(root) == 1:
            return root
        else:
            return None
```

669. Trim a Binary Search Tree
We will use recursion to return nodes within the boundaries.

```
class Solution:
    def trimBST(self, root: TreeNode, L: int, R: int) ->
TreeNode:
        if not root:
            return None

        if root.val < L:
            return self.trimBST(root.right, L, R)
        elif root.val > R:
            return self.trimBST(root.left, L, R)
        else:
            root.left = self.trimBST(root.left, L, R)
            root.right = self.trimBST(root.right, L, R)
            return root
```

112. Path Sum
We will traverse our tree in pre-order. TC is O(n)

```
class Solution:
    def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        def traverse(node, cur):
            if not node.left and not node.right:
                return cur + node.val == sum
            if node.left and traverse(node.left, cur + node.val):
                return True
            if node.right and traverse(node.right, cur +
node.val):
                return True
            return False
        if not root:
            return False
        return traverse(root, 0)
```

113. Path Sum II

We will use dfs to traverse all paths and append all qualified path to result. TC is O(n)

```
class Solution:
    def pathSum(self, root: TreeNode, sum: int) ->
List[List[int]]:
        def helper(node, cur, cur_arr, result):
            if not node.left and not node.right:
                if cur + node.val == sum:
                    result.append(cur_arr + [node.val])
            else:
                if node.left:
                    helper(node.left, cur + node.val, cur_arr +
[node.val], result)
                if node.right:
                    helper(node.right, cur + node.val, cur_arr +
[node.val], result)
        result = []
        if not root:
            return result
        helper(root, 0, [], result)
        return result
```