Contributor Name: Anonymous
Company Name:Amazon
Profile:Software Development Engineer (SDE)
Job Location:Bangalore
Applied[OnCampus/OffCampus]:off-campus

Round:1
Detailed Round Description
Online Coding Interview focused on Data Structures and Algorithms (DSA) and Object-Oriented Programming (OOPS).
Detailed Question Description(with Test Cases, if possible)

Problem 1: **Minimum Cost Path**
Description:
You are given an integer array `cost` where `cost[i]` is the cost of the `i-th` step on a staircase. Once you pay the cost, you can either climb one or two steps. You can start from the step with index `0` or `1`. Return the minimum cost to reach the top of the floor.

Problem Approach:
Used Dynamic Programming (DP) to solve this problem. Calculated the minimum cost to reach the `i-th` stair and stored it in the DP array, using it to find the cost for the `(n-1)-th` index.

Code Logic Block

nput: Array of integers `cost`
Output: Minimum cost to reach the top

1. **Initialize**:
   - `dp = [0] * len(cost)`
   - `dp[0] = cost[0]`
   - `dp[1] = cost[1]`
2. **Iterate through the array starting from index 2**:
   - `dp[i] = cost[i] + min(dp[i - 1], dp[i - 2])`
3. **Return the minimum cost of the last two stairs**:
   - `return min(dp[-1], dp[-2])`

Optimized Space Complexity:
Instead of maintaining a full `dp` array, use two variables:

1. `prev1 = cost[1]`
2. `prev2 = cost[0]`
   Update using the relation and keep the minimum of the last two.

Problem 2: **Sum Root to Leaf**

Description:

You are given the root of a binary tree containing digits from `0` to `9` only. Each root-to-leaf path in the tree represents a number. For example, the root-to-leaf path `1 -> 2 -> 3` represents the number `123`. Return the total sum of all root-to-leaf numbers.

Code Logic Block

Input: Root of the binary tree
Output: Total sum of all root-to-leaf numbers

1. **Helper Function**:
   - Define a recursive helper function `dfs(node, current_sum)`
   - If the node is `None`: Return `0`
   - Update the current sum: `current_sum = current_sum * 10 + node.val`
2. **Check Leaf Node**:
   - If it's a leaf node (`node.left == None` and `node.right == None`): Return `current_sum`
3. **Recurse**:
   - Return the sum of the left and right subtrees:
     - `return dfs(node.left, current_sum) + dfs(node.right, current_sum)`
4. **Call the Helper**:
   - Return `dfs(root, 0)`
   -

Problem 3: System Design (OOPS Concepts)
Description:
Focus on Object-Oriented Programming (OOPS) concepts with implementation. Asked to implement polymorphism, friend functions, and inheritance.