Contributor Name: Anonymous
Company Name: JP Morgan
Profile/Role: SDE - 1
Job Location: [Not Specified]
Applied (OnCampus/OffCampus): OnCampus

Round 01: Interview Round – Medium (Video Call)

Detailed Round Description:
This round featured two coding problems aimed at testing fundamental coding skills and problem-solving abilities.

Problem 1: Convert Binary Tree to Mirror Tree

Description:
Given a binary tree, convert this binary tree into its mirror tree. A binary tree is defined as a tree in which each parent node has at most two children

Mirror of a Tree: For every non-leaf node, interchange the left and right children.

Note: Modify the tree in-place.

Problem Approach:
Traverse the binary tree recursively. For each node, first recursively call the mirror function on the left and right subtrees and then swap the left and right children using a temporary variable.

Code Logic Block

Input: A binary tree (each node with up to two children)

Output: The mirror image of the input binary tree (in-place modification)

Key Steps:

- **Initialization: Begin at the root node.**
- **Main Logic:**
  1. **Call mirror function for the left subtree.**
  2. **Call mirror function for the right subtree.**
  3. **Swap the left and right subtrees:**
     - `temp = left-subtree`
     - `left-subtree = right-subtree`
     - `right-subtree = temp`
- **Return/Final Step: The root node now represents the mirror tree.**

Optimizations/Additional Notes:

Ensure that the swapping is done in-place to avoid extra space usage.

Problem 2: Palindrome Partitioning

Description:
Given a string 'S', partition it such that every substring of the partition is a palindrome. For example, for the string “BaaB”, the possible palindrome partitions are:

- {“B”, “a”, “a”, “B”}
- {“B”, “aa”, “B”}
- {“BaaB”}
  Each partition ensures that every substring is a palindrome.

Code Logic Block

Input: A string 'S'

Output: All possible partitions of the string such that every substring is a palindrome

Key Steps:

- Initialization: Set up recursion/backtracking structures.
- Main Logic:
  Use a recursive/backtracking approach to partition the string and check for palindromic substrings at each step.
- Return/Final Step: Return the list of all valid palindrome partitionings.

Optimizations/Additional Notes:
Consider memoization if overlapping subproblems are detected.

Round 02: Video Call

Detailed Round Description:
This round featured 2 coding problems that assessed problem-solving skills using data structures and algorithm logic,it lasted 60 mins

Problem 1:Bottom View Of Binary Tree

Description:
Given a binary tree, return the bottom view of the tree. The bottom view consists of the bottom-most nodes at each horizontal distance from the root.
Note:

- The horizontal distance for the root is 0.
- Left child: horizontal distance -1; right child: horizontal distance +1.
- If multiple nodes share the same horizontal distance, the node that is lower and more to the right is chosen.

Problem Approach:
Perform a level order traversal (using a queue) while tracking the horizontal distance for each node. Use a map (or dictionary) to store/update nodes at each horizontal distance ensuring the bottom-most node remains.

Code Logic Block

Input: A binary tree

Output: A list representing the bottom view of the binary tree

Key Steps:

- **Initialization:** Start with the root node at horizontal distance 0.
- **Main Logic:**
  1. Use a queue for level order traversal.
  2. For each node, record/update its horizontal distance in the map.
  3. Enqueue left child with hd-1 and right child with hd+1.
- **Return/Final Step:** Traverse the sorted keys of the map and output their corresponding node values.

Optimizations/Additional Notes:
Ensure that nodes at the same horizontal distance are updated correctly to reflect the bottom-most element.

Problem 2: Anagram Pairs

Description:

Given two strings 'str1' and 'str2', determine whether they form an anagram pair. Two strings form an anagram pair if the letters of one string can be rearranged to form the other.

Example: “spar” and “rasp” are anagrams.

Problem Approach:

Utilize hashing to compare the frequency of characters in both strings.

Code Logic Block

Input: Two strings, 'str1' and 'str2'

Output: A boolean indicating whether the strings form an anagram pair

Key Steps:

- **Initialization: Create a frequency map for each string.**
- **Main Logic:**
  Compare the frequency maps of the two strings.
- **Return/Final Step: Return `true` if both frequency maps are identical, else return `false`.**

Optimizations/Additional Notes:
This solution runs in O(n) time where n is the length of the strings.

Round 03: Video Call

Detailed Round Description:
This round focused on a single coding problem that involved transforming a data structure while maintaining traversal order., It lasted 60 mins

Problem 1:Flatten Binary Tree to Linked List

Description:

Given a binary tree consisting of 'n' nodes, convert it into a linked list in the same order as the pre-order traversal. The right pointer of the binary tree should serve as the "next" pointer for the linked list, and the left pointer should be set to NULL.

Note: Use the existing nodes only; do not create extra nodes.

Problem Approach:

Perform a pre-order traversal of the binary tree while re-linking nodes to form a linked list.

Code Logic Block

Input: A binary tree with integer values

Output: A linked list representing the pre-order traversal of the binary tree

Key Steps:

- **Initialization:** Start at the root node.
- **Main Logic:**
  Traverse the tree in pre-order while adjusting the pointers:
  1. Set the right pointer of the current node to point to the next node in the pre-order sequence.
  2. Set the left pointer to NULL.
- **Return/Final Step:** The modified root node now represents the head of the flattened linked list.

Optimizations/Additional Notes:
Ensure that the transformation is done in-place without allocating additional nodes.