Binary tree

Assignment Solutions

PW SKILLS

Q1. ASS_Code1.java

Approach:
A simple solution is to print all nodes of level 1 first, followed by level 2, … till level h, where h is the tree's height. We can print all nodes present in a level by modifying the preorder traversal on the tree.

We will keep on iterating at each level, starting from level 1. If the level is odd we will print left to right, else we will print right to left.

Time Complexity: O(N) as we are visiting every node once.

Q2. ASS_Code2.java

Approach:
- We can modify level order traversal to check if a given binary tree is a complete binary tree or not.
- The idea is for every dequeued node, check if it is a full node (have both left and right children).
- If a node is found that is not a full node, i.e., either it has no children or only one child, then all the remaining nodes in the queue should not have any children.
- If anyone has a child, then it's not a complete binary tree; otherwise, it is.

Time Complexity: O(N) as we are visiting every node once.

Q3. ASS_Code3.java

A simple solution would be to print all nodes of level h first, followed by level h-1, until level 1, where h is the tree's height. We can print all nodes present in a level by modifying the preorder traversal on the tree. The time complexity of this solution is O(n2), where n is the total number of nodes in the binary tree.

We can reduce the time complexity to O(n) by using extra space. Following is a pseudocode for a simple queue-based reverse level order traversal, which requires space proportional to the maximum number of nodes at a given depth. It can be as much as half of the total number of nodes.

```
levelorder(root)

q —> empty queue
s —> empty stack
q.enqueue(root)
while (not q.isEmpty())
  node —> q.dequeue()
  s.push(node)
  if (node.right <> null)
    q.enqueue(node.right)
  if (node.left <> null)
    q.enqueue(node.left)

while (not s.isEmpty())
  node —> s.pop()
  print(node)
```

Q4. ASS_Code4.java

We can also solve this problem by using hashing. The idea is to traverse the tree in a preorder fashion and pass level information in function arguments. If the level is visited for the first time, insert the current node and level information into the map. Finally, when all nodes are processed, traverse the map and print the left view.
Time Complexity: O(N) as we are visiting every node once.

Q5. ASS_Code5.java

The idea is simple – traverse the tree in a postorder fashion, and for every node, swap its left and right child pointer after recursively converting its left and right subtree to mirror first.