Binary Tree: BFS (Breadth-First Search) — Coding Interview Notes (Light Theme)

General Pattern Template

```
from collections import deque

def fn(root):
    queue = deque([root])
    ans = 0

    while queue:
        current_length = len(queue)
        # do logic for current level

        for _ in range(current_length):
            node = queue.popleft()
            # do logic
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

    return ans
```

Concept:
The **Breadth-First Search (BFS)** pattern, also known as *level-order traversal*, explores nodes level by level using a queue. Each iteration processes all nodes at the current depth before moving to the next.

Typical Uses: Level-based aggregation, shortest path or depth calculations, tree serialization/deserialization.
Time Complexity: O(n) **Space Complexity:** O(w), where w is the maximum width of the tree.

Key Ideas

1. Use a queue to track the current level nodes.
2. Each iteration of the while-loop processes one level.
3. Track level size before iterating, to handle per-level logic cleanly.
4. BFS naturally produces level-order traversal results (lists of lists).
5. Useful for shortest path problems or when ordering by depth matters.

Example 1: Level Order Traversal

Goal: Return a list of values for each level from top to bottom.
Approach: Process each level fully before enqueuing the next level.

```
from collections import deque

def level_order(root):
    if not root:
        return []
    queue = deque([root])
    result = []

    while queue:
        level = []
        for _ in range(len(queue)):
            node = queue.popleft()
            level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        result.append(level)

    return result
```

Example 2: Maximum Depth of Binary Tree

Goal: Compute the maximum number of levels (height) of the tree.
Approach: Use BFS and increment depth after each level processed.

```
from collections import deque

def max_depth(root):
    if not root:
        return 0
    queue = deque([root])
    depth = 0

    while queue:
        for _ in range(len(queue)):
            node = queue.popleft()
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        depth += 1

    return depth
```

Example 3: Average of Each Level

Goal: Compute the average node value at each level.
Approach: Sum node values per level and divide by level size.

```
from collections import deque

def average_of_levels(root):
    if not root:
        return []
    queue = deque([root])
    result = []

    while queue:
        level_sum = 0
        size = len(queue)
        for _ in range(size):
            node = queue.popleft()
            level_sum += node.val
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        result.append(level_sum / size)

    return result
```

Example 4: Minimum Depth of Binary Tree

Goal: Find the shortest path from root to any leaf node.
Approach: Stop BFS when the first leaf is found.

```
from collections import deque

def min_depth(root):
    if not root:
        return 0
    queue = deque([(root, 1)])

    while queue:
        node, depth = queue.popleft()
        if not node.left and not node.right:
            return depth
        if node.left:
            queue.append((node.left, depth + 1))
        if node.right:
            queue.append((node.right, depth + 1))
```

Summary Table

ProblemLogic per LevelUse CaseComplexity Level Order TraversalCollect valuesPrint tree by levelsO(n) Max DepthIncrement after each levelCompute tree heightO(n) Average of LevelsSum and divide per levelStatistical aggregationO(n) Min DepthReturn when first leaf foundShortest root-to-leaf

pathO(n)