Alternate Extreme Nodes Traversal

Objective: Return nodes of a binary tree at each level that are at the extreme corners (leftmost or rightmost) in an alternating order.

Algorithm Steps:

Step 1: Use BFS Level Order Traversal

- We can use a queue to traverse the tree level by level.
- Standard level order traversal using a queue.

Step 2: Track Level Size

- For each level, get the number of nodes using `q.size()`.
- This helps us identify first and last nodes at the current level.

Step 3: Alternate Extreme Selection

- Maintain a boolean `rightTaken` to alternate between rightmost and leftmost selection.
- If `rightTaken == 0`: pick **rightmost node** of current level.
- If `rightTaken == 1`: pick **leftmost node** of current level.
- Flip `rightTaken` at the end of each level.

Step 4: Process Nodes in Level

- For each node in the level:
- Push its left and right children into the queue.
- If current node is an extreme (based on `rightTaken`), append to the result vector.

Step 5: Return Result

- After processing all levels, return the result vector containing alternating extreme nodes.

Pseudo Code:

```
function extreme(Result, root):
    initialize queue q
    push root to q
    rightTaken = false

    while q is not empty:
        size = q.size()

        for i = 0 to size-1:
            temp = q.front()
            q.pop()
            if not rightTaken and i == size-1:
                append temp.data to Result
            if rightTaken and i == 0:
                append temp.data to Result

            if temp.left exists: push temp.left to q
            if temp.right exists: push temp.right to q

        rightTaken = !rightTaken

function extremeNodes(root):
    initialize Result as empty vector
    call extreme(Result, root)
    return Result
```

Notes:

- `rightTaken` alternates at each level to select extremes in alternating order.
- Queue ensures we process nodes level by level.
- Leftmost or rightmost node is picked by checking loop index.

Time Complexity: O(n) # Each node is visited once. **Space Complexity:** O(n) # For queue storage of nodes.