Binary Tree: Printing Elements at Nth Level

Problem Statement

Given a binary tree, print all the elements present at the nth level of the tree. The level of the root node is considered as 0. The function should print the nodes level-wise in a structured format.

Understanding the Binary Tree Representation

A binary tree consists of nodes where each node has:

- A value (val)
- A pointer to the left child (left)
- A pointer to the right child (right)

Each level in the binary tree refers to the depth of nodes from the root.

For example, consider the binary tree:

```
     1
    / \
   2   3
  / \ / \
 4  5 6  7
```

- Level 0: 1
- Level 1: 2, 3
- Level 2: 4, 5, 6, 7

Code Explanation

The following C++ program constructs a binary tree and prints its elements level-wise:

```
#include<iostream>
#include <climits>
using namespace std;

class node{
 public:
 int val;
 node* right;
```

```
  node* left;
  node(int val){
    this->val = val;
    right = NULL;
    left = NULL;
  }
};

// Function to calculate the height (or levels) of the binary tree
int level(node* root){
  if(root == NULL) return 0;
  return 1 + max(level(root->left),level(root->right));
}

// Function to print all nodes at a given level
void nth_level_display(node* root, int lvl, int t_lvl){
  if(root == NULL) return;
  if(lvl == t_lvl){
    cout << root->val << " ";
    return;
  }
  nth_level_display(root->left, lvl+1, t_lvl);
  nth_level_display(root->right, lvl+1, t_lvl);
}

// Function to print the binary tree level-wise in a structured format
void level_wise_display(node* root){
  int n = level(root);
  for(int i=0; i<n; i++){
    for(int j=n-1-i; j>0; j--){
      cout << " ";  // Indentation for better visualization
```

```
    }
    nth_level_display(root, 0, i);
    cout << endl;
  }
}

int main(){
  // Creating nodes of the binary tree
  node* a = new node(1);
  node* b = new node(2);
  node* c = new node(3);
  node* d = new node(4);
  node* e = new node(5);
  node* f = new node(6);
  node* g = new node(7);

  // Constructing the tree
  a->left = b;
  a->right = c;
  b->left = d;
  b->right = e;
  c->left = f;
  c->right = g;

  // Display tree level-wise
  level_wise_display(a);

  return 0;
}
```

Explanation of Code Components

1. **Class Definition (node)**
    - Defines the structure of a tree node with left and right child pointers.
    - Constructor initializes the node's value and sets left and right children to NULL.
2. **level() Function**
    - Recursively calculates the height (or maximum depth) of the tree.
    - Uses max() to find the height of left and right subtrees and adds 1 for the root.
3. **nth_level_display() Function**
    - Recursively prints all nodes at a given level t_lvl.
    - Base case: If lvl == t_lvl, print the node's value.
    - Recursively calls left and right subtrees, increasing lvl.
4. **level_wise_display() Function**
    - Determines the height n of the tree using level().
    - Iterates through all levels and calls nth_level_display() for each level.
    - Uses spaces for indentation to represent tree structure visually.
5. **main() Function**
    - Creates a binary tree manually.
    - Calls level_wise_display() to print the elements level-wise.

Output of the Program

1

2 3

4 5 6 7

This output correctly represents the tree structure in a level-wise manner.

Time Complexity Analysis

- level() function: **O(N)** (since it traverses all nodes to find the height)
- nth_level_display() function: **O(N)** in worst case per level.
- level_wise_display() calls nth_level_display() O(logN) times, making it **O(N logN)** in the worst case.

For better performance, **BFS (Breadth-First Search) using a queue** could be used to print levels in **O(N)**.

Improvements and Alternative Approaches

1. **Using BFS (Level Order Traversal with Queue)**
    - This approach can print levels in **O(N)** time complexity.
    - Uses a queue to process each level iteratively.
2. **Dynamic Tree Construction**
    - Instead of manually assigning nodes, take input dynamically.
3. **Reverse Level Order Traversal**
    - Can be achieved using a queue and a stack.

Conclusion

- The given program efficiently prints elements at each level.
- The approach uses **recursive traversal**, which is simple but can be optimized.
- Alternative approaches like **BFS using a queue** can improve performance for large trees.

This explanation provides a complete understanding of the problem and solution. 🚀