Assignment 4

Course: CS_413 - Analysis of Algorithms
Assignment: 4
Date: 04/10/2025
Group Member(s): 1) Aryan Jigneshbhai Bhagat (sl5310)
2) Moksha Kumudbhai Shah (bp4199)

Problem Statement:

We are given the root of a binary tree. A node at level 0 is considered to be at an **even level**. The **root node** is at level 0, its children are at level 1, its grandchildren are at level 2, and so forth. Our goals are:

1) **Implement a recursive program** (in C++ or Java) that counts the number of nodes at **even levels** of a binary tree.
2) **Demonstrate** how this program can handle an **arbitrary binary tree** input.
3) **Analyze** the **time complexity** of the program in the worst case (when there are n nodes in the tree). Provide the **recurrence relation** and derive the **Big-O** notation

Solution:

1) We use a **recursive traversal** of the binary tree.
2) Each time we visit a node, we check its **level**:
    a) If (level % 2 == 0), we increment the count because it is at an even level (e.g. levels 0, 2, 4, ...).
    b) We then recursively visit both the left and right children, passing an incremented level.

Hence, the **count** of nodes at even levels can be computed by combining (summing) the results from each subtree plus the contribution of the current node if it is at an even level.

Algorithmic Steps:

1) **Base Case:** If the current node is NULL (i.e., root == nullptr), return 0.
2) **Check Even Level:** If the level is even, add 1 to the count; otherwise, add 0.

3) **Recurse:** Recursively call the function on:
    a) root->left with level + 1
    b) root->right with level + 1
4) **Combine:** Sum these values:

count = (isEvenLevel ? 1 : 0) + leftSubtreeCount + rightSubtreeCount

Time Complexity Analysis:

Let n be the number of nodes in the binary tree. We traverse each node exactly once in a preorder-like manner (checking the node, then recursing on its children).

Recurrence Relation:

If T(n) denotes the time to process a tree of n nodes, the recursive structure indicates:

$$T(n) = T(n_l) + T(n_r) + O(1)$$

where n_l and n_r are the sizes of the left and right subtrees, respectively, and $n_l + n_r = n-1$. The additional O(1) work stems from checking if the current level is even and performing a constant-time addition.

Because we eventually visit each node once, summing up the work done across all recursive calls yields:

$$T(n)=O(n)$$

In other words, each node contributes a constant amount of work, so the total work is linear in the number of nodes.

Edge Cases:

- An empty tree (e.g., n = 0) leads to root == nullptr, and the result is `0`.
- A tree where some nodes have only one child is handled naturally by the code.

Worst-Case Time Complexity: **O(n)** for traversing n nodes exactly once.

C++ Code Implementation:

```cpp
// Submitted by : Aryan Jigneshbhai Bhagat - NetID: sl5310, & Moksha Kumudbhai Shah - NetID: bp4199

// CS_411 - Assignment 4 - C++ program to count nodes at even levels in a binary tree

#include<bits/stdc++.h>
using namespace std;

// Binary tree node structure
struct TreeNode {
    string val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(string x) : val(x), left(nullptr), right(nullptr) {}
};

// Recursive function to count nodes at even levels
int countEvenLevelNodes(TreeNode* root, int level = 0) {
    if (!root) return 0;
    int count = (level % 2 == 0) ? 1 : 0;
    return count + countEvenLevelNodes(root->left, level + 1) + countEvenLevelNodes(root->right, level + 1);
}

// Build tree from input (child->parent format)
TreeNode* buildTree(const vector<tuple<string, string, string>>& input) {
    unordered_map<string, TreeNode*> nodes;
    unordered_map<string, bool> isChild;

    for (const auto& [parent, left, right] : input) {
        if (!nodes[parent]) nodes[parent] = new TreeNode(parent);
        if (!left.empty()) {
            nodes[left] = new TreeNode(left);
            nodes[parent]->left = nodes[left];
```

```
            isChild[left] = true;
        }
        if (!right.empty()) {
            nodes[right] = new TreeNode(right);
            nodes[parent]->right = nodes[right];
            isChild[right] = true;
        }
    }

    // Find the root (not a child of any node)
    for (const auto& [val, node] : nodes) {
        if (!isChild[val]) return node;
    }
    return nullptr;
}

// Example usage
int main() {
    cout << "Enter number of nodes: ";
    int n;
    cin >> n;
    cin.ignore();

    vector<tuple<string, string, string>> input;
    cout << "Enter nodes in format: Parent LeftChild RightChild (use
'-' for null)\n";
    for (int i = 0; i < n; ++i) {
        string line;
        getline(cin, line);
        stringstream ss(line);
        string p, l, r;
        ss >> p >> l >> r;
        if (l == "-") l = "";
        if (r == "-") r = "";
        input.emplace_back(p, l, r);
    }

    TreeNode* root = buildTree(input);
    int count = countEvenLevelNodes(root);
```

```
    cout << "Count of nodes at even levels: " << count << endl;

    return 0;
}
```

README file:

```
Input Format Instructions:

The program expects the user to input the number of nodes and then
input each node's structure
in the format: Parent -> LeftChild -> RightChild

- If a node does not have a left or right child, use '-' (dash) as a
placeholder.
- Example input:
    3
    A B C
    B D -
    C - E

This represents:
        A (level 0)
       / \
      B   C (level 1)
     /     \
    D       E (level 2)

Even level nodes are:
- level 0: A
- level 2: D, E

The program will then compute the number of nodes at even levels (0,
2, 4, ...).
```