Problem Name: Nearest Left Node in a Binary Tree

Problem Statement:

Given a binary tree and a target node, your task is to find the nearest left node on the same level as the target node. If the target node is the leftmost node on its level, return `-1`.

You are given the root of a binary tree and a reference to a node `u` in the tree. Your task is to find and return the nearest node to the left of `u` on the same level. If no such node exists (i.e., `u` is the leftmost node on its level), return `-1`.

Brute Force Approach:

1. Traverse the tree using level order traversal.
2. At each level, store the nodes in an array or list.
3. For the level where the target exists, find the node and return the node to its left, if any.
4. Time complexity: `O(n)` space and time.

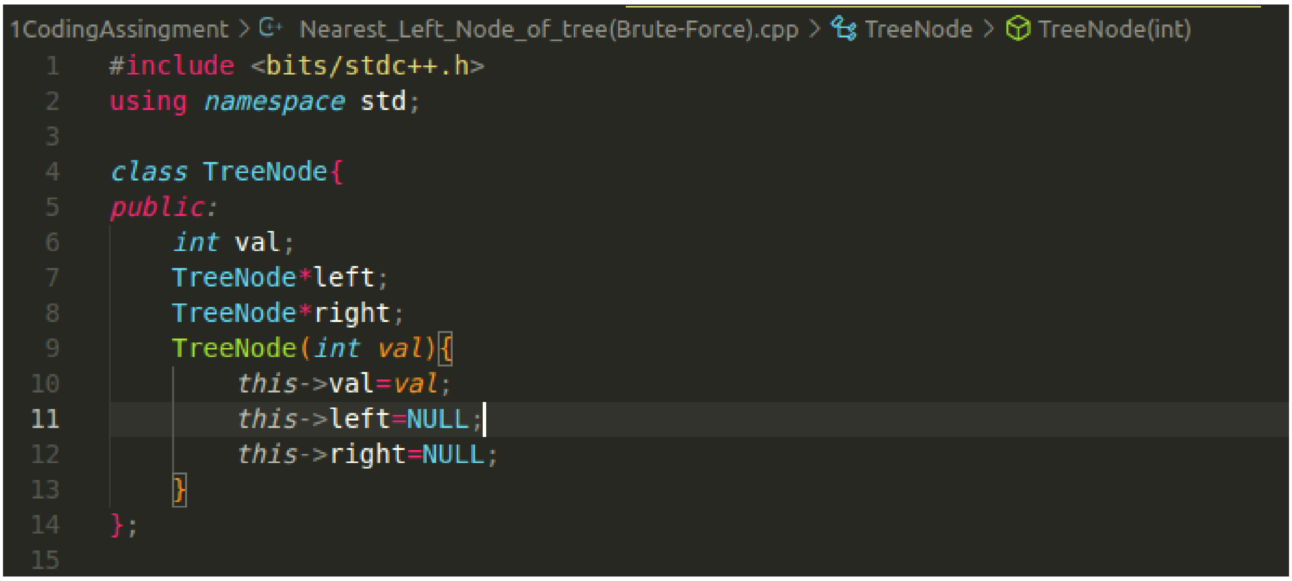
Time Complexity:

- **O(n)** — where `n` is the number of nodes in the tree (due to level-order traversal).

Space Complexity:

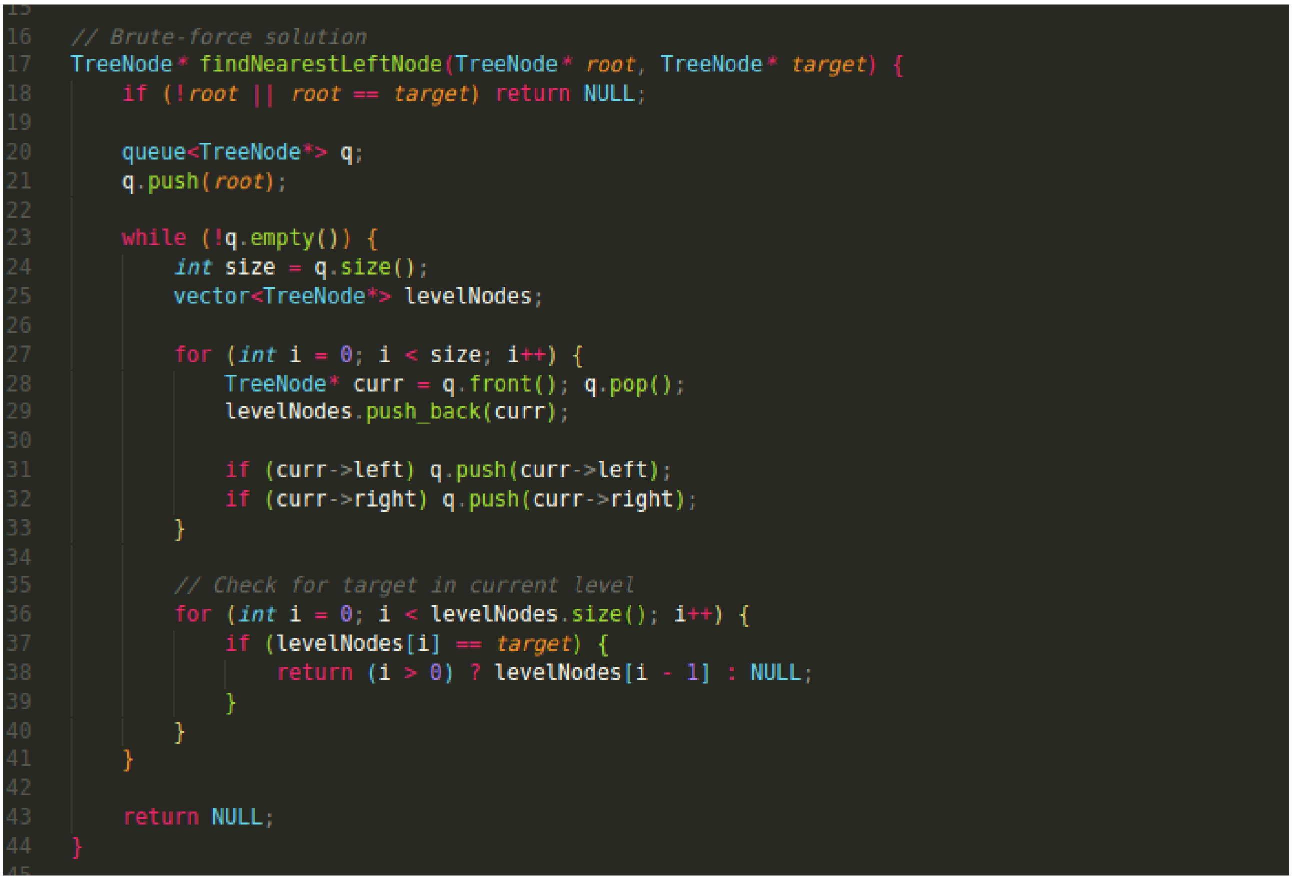
- **O(n)** — for storing nodes at each level and for the queue used in traversal.

Code

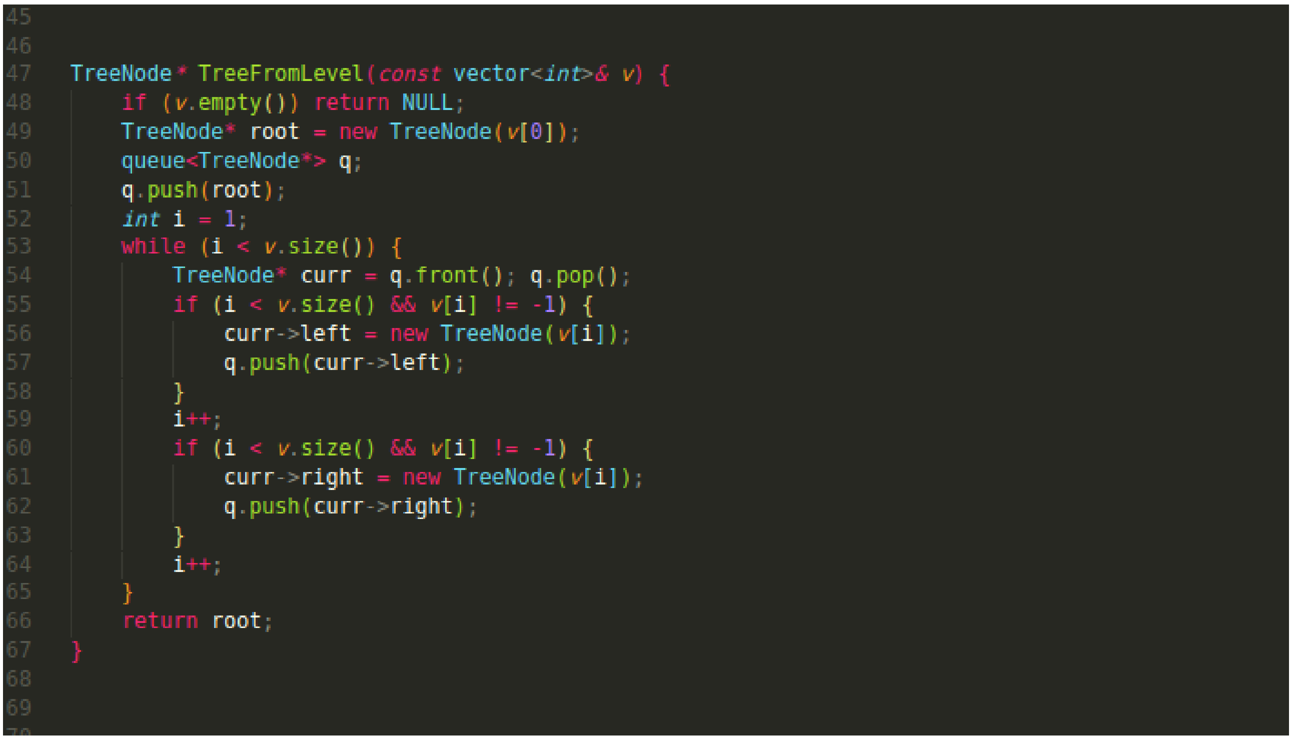


1CodingAssingment > Nearest_Left_Node_of_tree(Brute-Force).cpp > TreeNode > TreeNode(int)

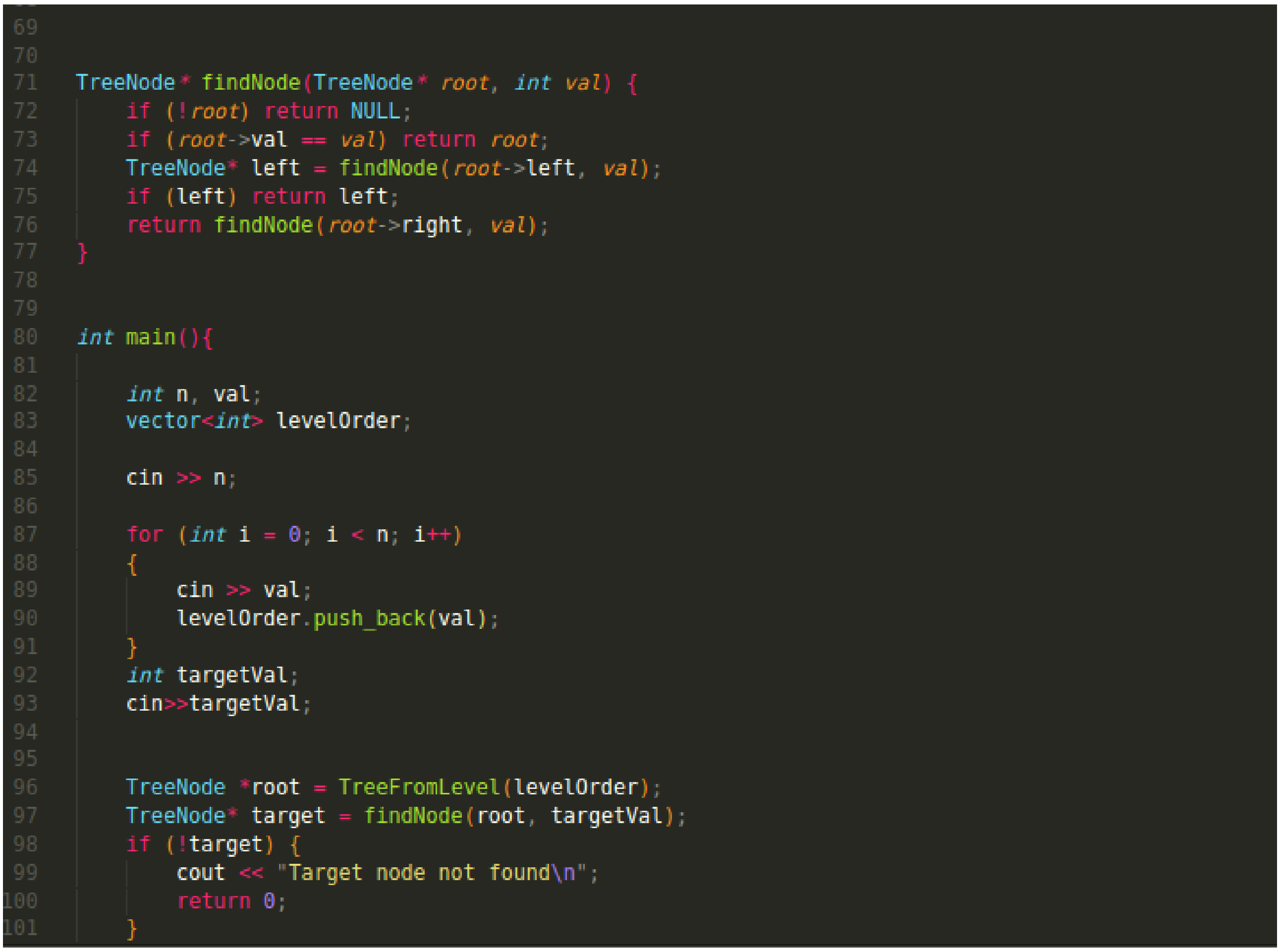
```
#include <bits/stdc++.h>
using namespace std;

class TreeNode{
public:
    int val;
    TreeNode*left;
    TreeNode*right;
    TreeNode(int val){
        this->val=val;
        this->left=NULL;
        this->right=NULL;
    }
};

```



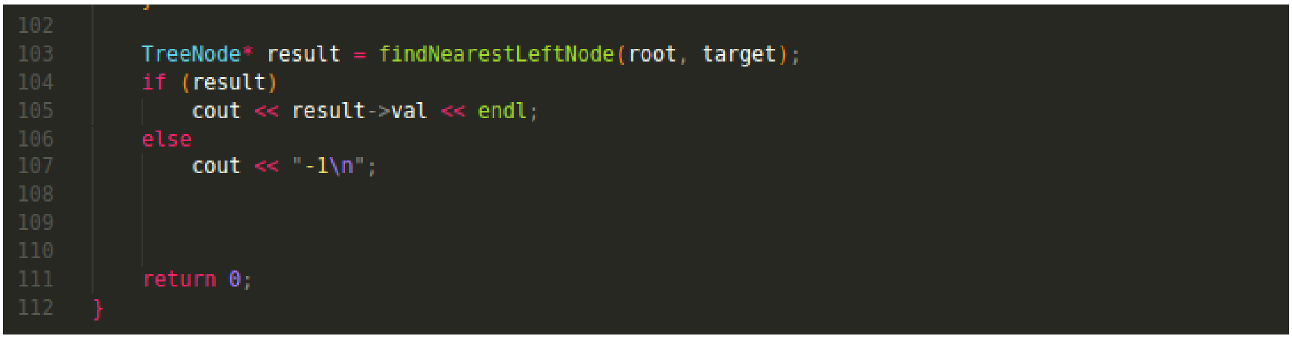
```cpp
// Brute-force solution
TreeNode* findNearestLeftNode(TreeNode* root, TreeNode* target) {
    if (!root || root == target) return NULL;

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int size = q.size();
        vector<TreeNode*> levelNodes;

        for (int i = 0; i < size; i++) {
            TreeNode* curr = q.front(); q.pop();
            levelNodes.push_back(curr);

            if (curr->left) q.push(curr->left);
            if (curr->right) q.push(curr->right);
        }

        // Check for target in current level
        for (int i = 0; i < levelNodes.size(); i++) {
            if (levelNodes[i] == target) {
                return (i > 0) ? levelNodes[i - 1] : NULL;
            }
        }
    }

    return NULL;
}
```



```cpp
TreeNode* TreeFromLevel(const vector<int>& v) {
    if (v.empty()) return NULL;
    TreeNode* root = new TreeNode(v[0]);
    queue<TreeNode*> q;
    q.push(root);
    int i = 1;
    while (i < v.size()) {
        TreeNode* curr = q.front(); q.pop();
        if (i < v.size() && v[i] != -1) {
            curr->left = new TreeNode(v[i]);
            q.push(curr->left);
        }
        i++;
        if (i < v.size() && v[i] != -1) {
            curr->right = new TreeNode(v[i]);
            q.push(curr->right);
        }
        i++;
    }
    return root;
}
```



```cpp


TreeNode* findNode(TreeNode* root, int val) {
    if (!root) return NULL;
    if (root->val == val) return root;
    TreeNode* left = findNode(root->left, val);
    if (left) return left;
    return findNode(root->right, val);
}


int main(){

    int n, val;
    vector<int> levelOrder;

    cin >> n;

    for (int i = 0; i < n; i++)
    {
        cin >> val;
        levelOrder.push_back(val);
    }
    int targetVal;
    cin>>targetVal;


    TreeNode *root = TreeFromLevel(levelOrder);
    TreeNode* target = findNode(root, targetVal);
    if (!target) {
        cout << "Target node not found\n";
        return 0;
    }
```



```cpp

    TreeNode* result = findNearestLeftNode(root, target);
    if (result)
        cout << result->val << endl;
    else
        cout << "-1\n";



    return 0;
}
```

Approach (Optimized Brute Force using BFS)

1. Perform a level-order traversal using a queue.
2. For each level, track the `prev` node (i.e., the last node visited in the current level).
3. If the current node is the target `u`, return `prev`.
4. At each level, before moving to the next node, update `prev = curr`.
5. If no left neighbor is found, return `NULL`.

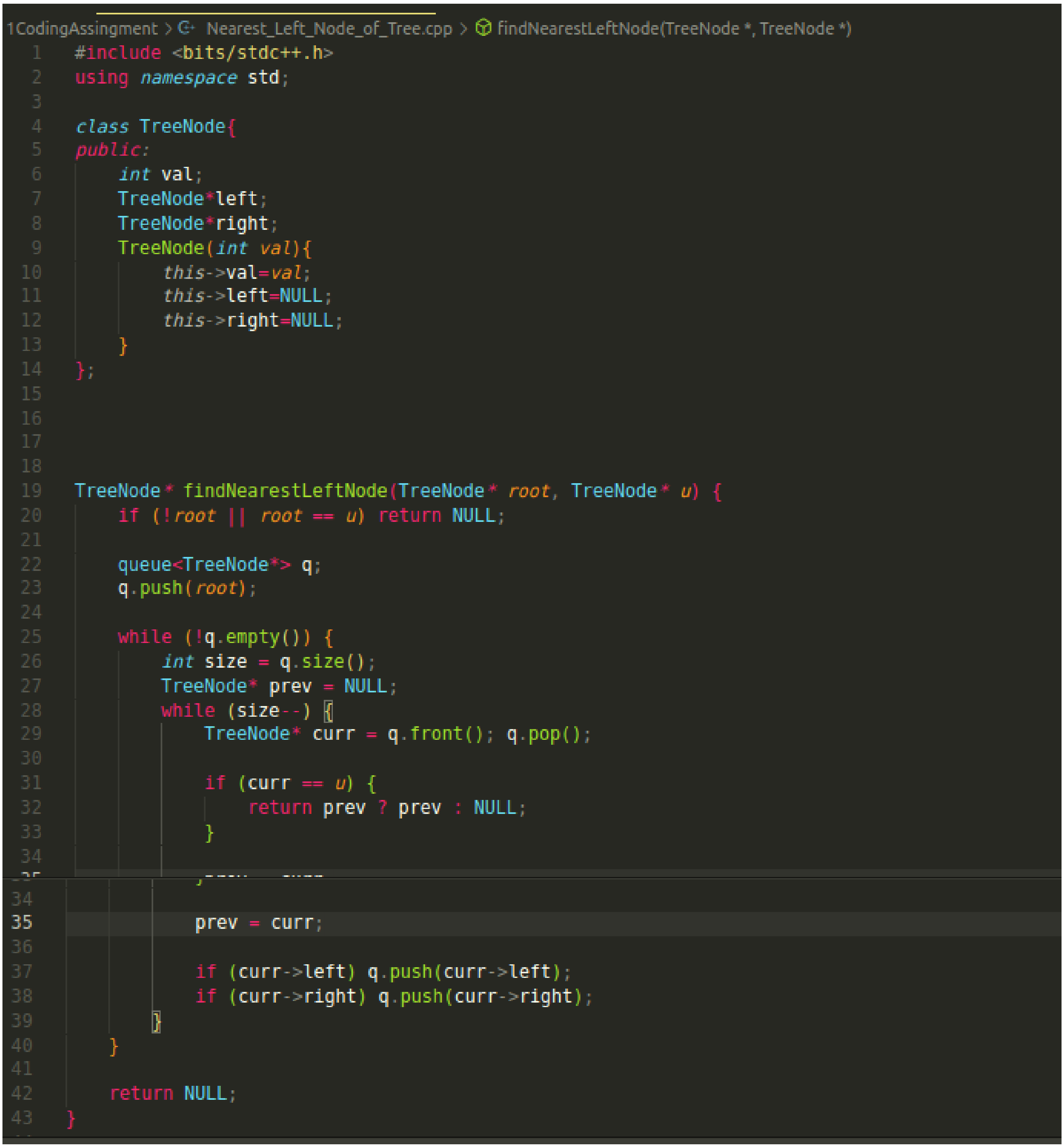
Time Complexity

- **O(n)** — where n is the number of nodes in the tree (due to level-order traversal).

Space Complexity

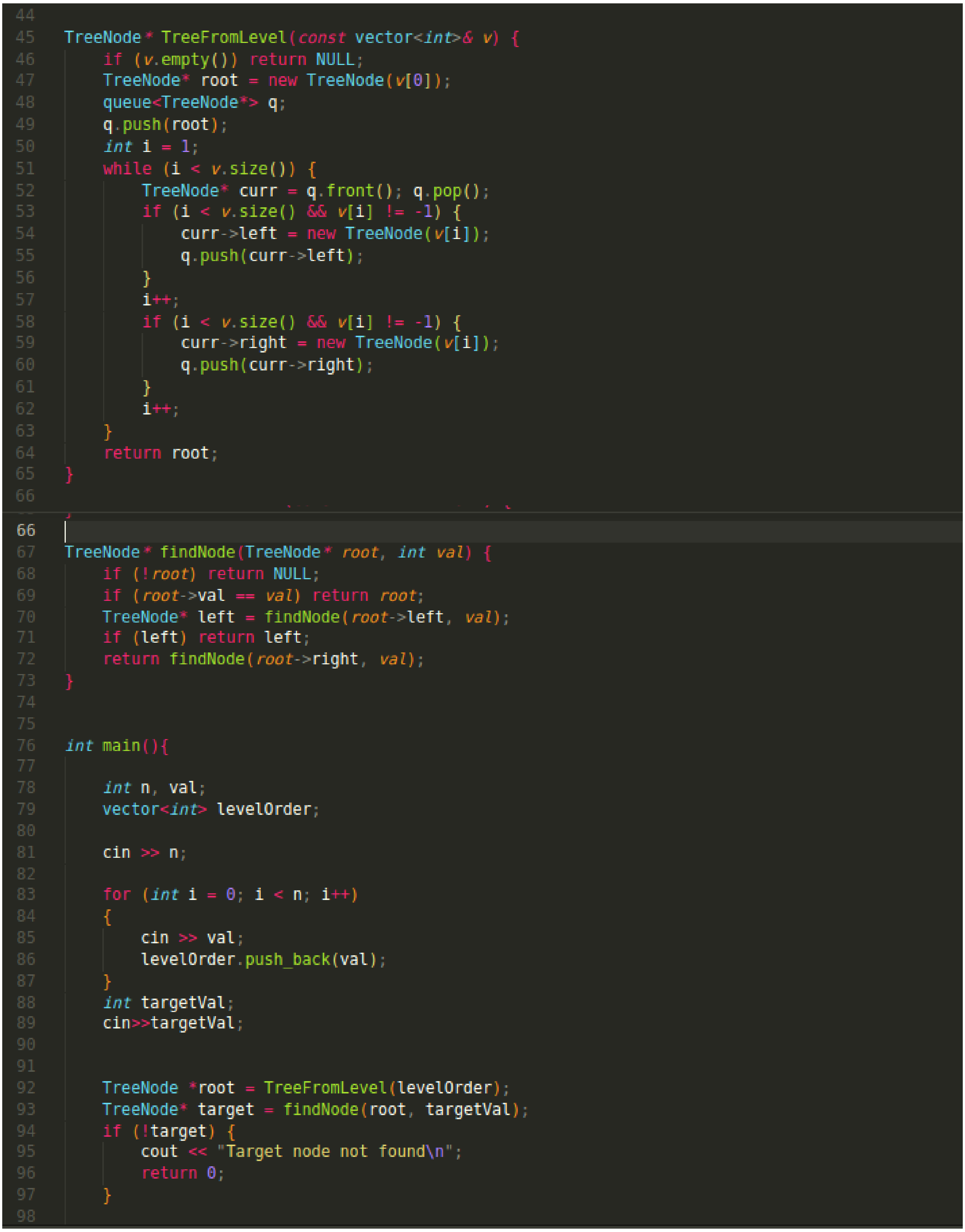
- **O(1)** – As we are not Storing Any Element in array

Code



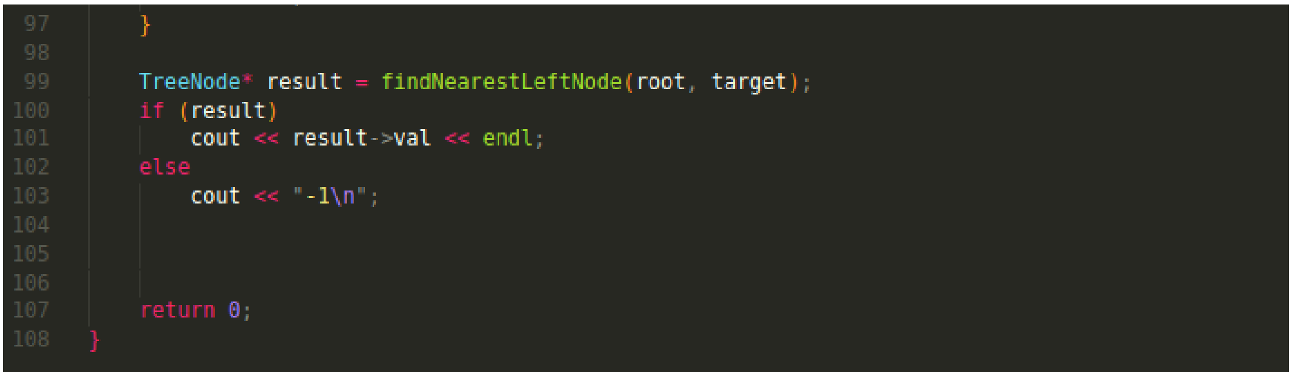
1CodingAssingment > Nearest_Left_Node_of_Tree.cpp > findNearestLeftNode(TreeNode *, TreeNode *)

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

class TreeNode{
public:
    int val;
    TreeNode*left;
    TreeNode*right;
    TreeNode(int val){
        this->val=val;
        this->left=NULL;
        this->right=NULL;
    }
};




TreeNode* findNearestLeftNode(TreeNode* root, TreeNode* u) {
    if (!root || root == u) return NULL;

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int size = q.size();
        TreeNode* prev = NULL;
        while (size--) {
            TreeNode* curr = q.front(); q.pop();

            if (curr == u) {
                return prev ? prev : NULL;
            }

            prev = curr;

            if (curr->left) q.push(curr->left);
            if (curr->right) q.push(curr->right);
        }
    }

    return NULL;
}
```



```cpp

TreeNode* TreeFromLevel(const vector<int>& v) {
    if (v.empty()) return NULL;
    TreeNode* root = new TreeNode(v[0]);
    queue<TreeNode*> q;
    q.push(root);
    int i = 1;
    while (i < v.size()) {
        TreeNode* curr = q.front(); q.pop();
        if (i < v.size() && v[i] != -1) {
            curr->left = new TreeNode(v[i]);
            q.push(curr->left);
        }
        i++;
        if (i < v.size() && v[i] != -1) {
            curr->right = new TreeNode(v[i]);
            q.push(curr->right);
        }
        i++;
    }
    return root;
}

```

```cpp

TreeNode* findNode(TreeNode* root, int val) {
    if (!root) return NULL;
    if (root->val == val) return root;
    TreeNode* left = findNode(root->left, val);
    if (left) return left;
    return findNode(root->right, val);
}


int main(){

    int n, val;
    vector<int> levelOrder;

    cin >> n;

    for (int i = 0; i < n; i++)
    {
        cin >> val;
        levelOrder.push_back(val);
    }
    int targetVal;
    cin>>targetVal;


    TreeNode *root = TreeFromLevel(levelOrder);
    TreeNode* target = findNode(root, targetVal);
    if (!target) {
        cout << "Target node not found\n";
        return 0;
    }

```



```cpp
    }

    TreeNode* result = findNearestLeftNode(root, target);
    if (result)
        cout << result->val << endl;
    else
        cout << "-1\n";



    return 0;
}
```