**Problem Name:** Binary Tree Tilt

**Topics:** DFS, Tree, Binary Tree

**Companies:** Indeed

**Level:** Easy

**Language:** C++

**Problem Statement:** Create a binary tree by taking input and -1 as null, return *the sum of every tree node's****tilt****.*

The **tilt** of a tree node is the **absolute difference** between the sum of all left subtree node **values** and all right subtree node **values**. If a node does not have a left child, then the sum of the left subtree node **values** is treated as 0. The rule is similar if the node does not have a right child.

**Input Format:** First line of input is n (total no of nodes in the tree)

Next line contains n space-separated integers representing values if each node and -1 as NULL

**Output Format:** Print sum of every tree node's tilt

**Constraints:**

* The number of nodes in the tree is in the range [0, 104].
* 0 <= Node.val <= 1000

**Examples:**

**Input:** root = [4,2,9,3,5,null,7]

**Output:** 15

**Explanation:**

Tilt of node 3 : |0-0| = 0 (no children)

Tilt of node 5 : |0-0| = 0 (no children)

Tilt of node 7 : |0-0| = 0 (no children)

Tilt of node 2 : |3-5| = 2 (left subtree is just left child, so sum is 3; right subtree is just right child, so sum is 5)

Tilt of node 9 : |0-7| = 7 (no left child, so sum is 0; right subtree is just right child, so sum is 7)

Tilt of node 4 : |(3+5+2)-(9+7)| = |10-16| = 6 (left subtree values are 3, 5, and 2, which sums to 10; right subtree values are 9 and 7, which sums to 16)

Sum of every tilt : 0 + 0 + 0 + 2 + 7 + 6 = 15

**Brute force Solution:**

**Explanation:**

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

    int data;

    Node \*left, \*right;

    Node(int val){

        data = val;

        left = nullptr;

        right = nullptr;

    }

};

int res = 0;

void find\_sum(Node\* root, int &sum)

{

    if(root == NULL)

        return ;

    sum = sum + root->data;

    find\_sum(root->left,sum);

    find\_sum(root->right,sum);

}

void helper(Node\* root)

{

    if(root == NULL)

        return ;

    int left\_sum = 0;

    int right\_sum = 0;

    find\_sum(root->left, left\_sum);

    find\_sum(root->right,right\_sum);

    res = res + abs(left\_sum - right\_sum);

    helper(root->left);

    helper(root->right);

}

int findTilt(Node\* root) {

    if(root == NULL)

        return 0;

    helper(root);

    return res;

}

int main()

{

    int n;

    cin >> n;

    vector<int> v(n);

    for (int i = 0; i < n; i++)

        cin >> v[i];

    queue<Node \*> q;

    Node \*root = new Node(v[0]);

    q.push(root);

    int i = 1;

    while (i < n)

    {

        Node \*curr = q.front();

        q.pop();

        if (curr)

        {

            Node \*temp1, \*temp2;

            if (v[i] != -1)

                temp1 = new Node(v[i]);

            else

                temp1 = nullptr;

            curr->left = temp1;

            q.push(temp1);

            if (i + 1 < n)

            {

                if (v[i + 1] != -1)

                    temp2 = new Node(v[i + 1]);

                else

                    temp2 = nullptr;

                curr->right = temp2;

                q.push(temp2);

            }

            i += 2;

        }

    }

    int result = findTilt(root);

    cout<<result;

    return 0;

}

**Time Complexity**: O(N2)

**Space Complexity:** O(height of tree)

**Optimized Solution:**

**Explanation:**

* The only thing we need to maintain is *subtree sum for every node* right? Why?
* Since **our answer is the summation of absolute values of the difference between left and right child subtree sum for every node**.
* Wait, How do we calculate the subtree sum and sum of tilt's value in one pass(one recursive step)?
* Maintain **pair<int, int>** for every node where *first element of the pair will be the sum of tilt's value for left child subtree and the second element of the pair will be left child subtree sum*.
* Now, **our answer will be the first element of the pair of the root node obviously**.
* Also, The key step is what do we return to the parents of every node?
* **return {l.first+r.first+abs(l.second-r.second),l.second+r.second+root->val}**
* You can easily observe that *first value is the sum of tilt's value of all nodes in the current subtree* and *the second element is subtree sum of the current node*.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

    int data;

    Node \*left, \*right;

    Node(int val){

        data = val;

        left = nullptr;

        right = nullptr;

    }

};

pair<int,int> BinaryTreeTilt(Node\* root){

    if(!root)

        return {0,0};

    pair<int,int> l = BinaryTreeTilt(root->left); // {sum of tilt values for left child,left child subtree sum}

    pair<int,int> r = BinaryTreeTilt(root->right); // {sum of tilt values for right child,right child subtree sum}

    return {l.first+r.first+abs(l.second-r.second),l.second+r.second+root->data};

}

int findTilt(Node\* root) {

    return BinaryTreeTilt(root).first; // answer is first element of pair

}

int main()

{

    int n;

    cin >> n;

    vector<int> v(n);

    for (int i = 0; i < n; i++)

        cin >> v[i];

    queue<Node \*> q;

    Node \*root = new Node(v[0]);

    q.push(root);

    int i = 1;

    while (i < n)

    {

        Node \*curr = q.front();

        q.pop();

        if (curr)

        {

            Node \*temp1, \*temp2;

            if (v[i] != -1)

                temp1 = new Node(v[i]);

            else

                temp1 = nullptr;

            curr->left = temp1;

            q.push(temp1);

            if (i + 1 < n)

            {

                if (v[i + 1] != -1)

                    temp2 = new Node(v[i + 1]);

                else

                    temp2 = nullptr;

                curr->right = temp2;

                q.push(temp2);

            }

            i += 2;

        }

    }

    int result = findTilt(root);

    cout<<result;

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(height of tree)