834. Sum of Distances in Tree

DescriptionHintsSubmissionsDiscussSolution

An undirected, connected tree with `N` nodes labelled `0...N-1` and `N-1` `edges` are given.

The `i`th edge connects nodes `edges[i][0]` and `edges[i][1]` together.

Return a list `ans`, where `ans[i]` is the sum of the distances between node `i` and all other nodes.

Example 1:

```
Input: N = 6, edges = [[0,1],[0,2],[2,3],[2,4],[2,5]]
Output: [8,12,6,10,10,10]
Explanation: 
Here is a diagram of the given tree:
  0
 / \
1   2
   /|\
  3 4 5
We can see that dist(0,1) + dist(0,2) + dist(0,3) + dist(0,4) + dist(0,5)
equals 1 + 1 + 2 + 2 + 2 = 8.  Hence, answer[0] = 8, and so on.
```

Note: `1 <= N <= 10000`

Seen this question in a real interview before?

- Subscribe to see which companies asked this question.

  Related Topics DFS

Approach #1: Subtree Sum and Count [Accepted]

Intuition and Algorithm

Root the tree. For each node, consider the subtree of that node plus all descendants, and consider `count[node]` and `stsum[node]`, the number of nodes and the sum of the value of those nodes.

We can calculate this using a post-order traversal, where on exiting some `node`, the `count` and `stsum` of all descendants of this node is correct, and we now calculate `count[node] += count[nei]` and `stsum[node] += stsum[nei] + count[nei]`.

This will give us the right answer for the `root`: `ans[root] = stsum[root]`.

Now for the insight: if we have a node `parent` and it's child `child`, then `ans[child] = ans[parent] - count[child] + (N - count[child])`. This is because there are `count[child]` nodes that are `1` easier to get to from `child` than `parent`, and `N-count[child]` nodes that are `1` harder to get to from `child` than `parent`.

Using a second, pre-order traversal, we can update our answer in linear time for all of our nodes.

```
void dfs1(int root, vector<unordered_set<int>> &tree, unordered_set<int>
&seen,

vector<int> &count, vector<int> &res)

{

    seen.insert(root);

    unordered_set<int> elem = tree[root];

    for(auto i:elem)

    {

        if(!seen.count(i))

        {

            dfs1(i,tree,seen,count,res);

            count[root]+=count[i];

            res[root]+=res[i]+count[i];

        }

    }

    count[root]+=1;

}


void dfs2(int root, vector<unordered_set<int>> &tree, unordered_set<int>
&seen,

vector<int> &count, vector<int> &res, int N)

{

    seen.insert(root);

    unordered_set<int> elem = tree[root];

    for(auto i:elem)

    {

        if(!seen.count(i))
```

```
        {
            res[i] = res[root] - count[i] + N - count[i];
            dfs2(i,tree,seen,count,res,N);
        }
    }
}

vector<int> sumOfDistancesInTree(int N, vector<vector<int>>& edges)
{
    vector<unordered_set<int>> tree(N);
    vector<int> res(N,0);
    vector<int> count(N,0);
    unordered_set<int> seen1,seen2;
    for(int i=0;i<(int)edges.size();i++)
    {
        int e1 = edges[i][0];
        int e2 = edges[i][1];
        tree[e1].insert(e2);
        tree[e2].insert(e1);

    }
    dfs1(0,tree,seen1,count,res);
    dfs2(0,tree,seen2,count,res,N);
    return res;
}
```