PRE-COMPUTATION USING DFS

Suppose, we have given $q = 10^5$ (queries) and then in each query we have to find given (n) subtree sum and even-count of that subtree.

∴ Simply, we will run a loop of q (queries) and inside each loop we will run 2 DFS, one for subtree sum and one for even count.

But, This will give TLE as

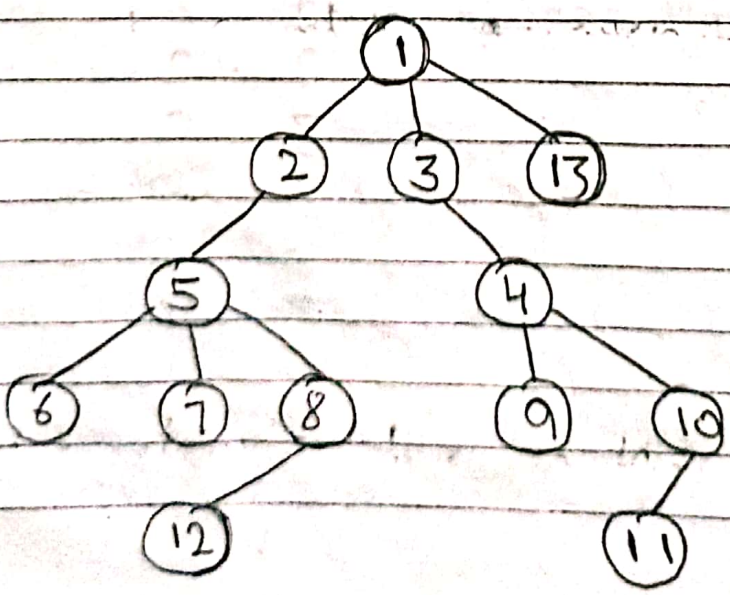
$q = 10^5$ & $(N \times M) \approx 10^7$

∴ $O((q)(N \times M)) \Rightarrow 10^{12}$ (TLE)

∴ We can prevent TLE simply using precomputation technique.

APPROACH: It will follow similar approach of finding height (h) of a tree.

e.g.:



	1	2	3	4	5	6	7	8	9	10	11	12	13
Subtree Sum (S.S)	91	40	37	33	38	6	7	20	9	21	11	12	13

NOTE Subtree sum of leaf node will be equal to its value.

CODE

```
-> Maximum capacity of nodes
CONST INT N = 1e5+10;
-> Adjacency list
VECTOR<INT> G[N];
-> To store pre-computed subtree sum
INT SUBTREE_SUM[N];
-> To store pre-computed even count of
 a subtree
INT EVEN_CT[N];

VOID DFS (INT VERTEX, INT PAR=0)
{
    -> Adding vertex's own value first
    SUBTREE_SUM[VERTEX] += VERTEX;
    -> Incrementing vertex's count if it
 itself is even
    IF (VERTEX % 2 == 0) EVEN_CT[VERTEX]++;

    FOR (INT CHILD : G[VERTEX])
    {
        IF (CHILD == PAR) CONTINUE ; //
        skipping visited node (parent)
        DFS (CHILD, VERTEX);
```

```
            -> Pre computing subtree sum
            SUBTREE_SUM [VERTEX] += SUBTREE_SUM [
                                          CHILD];
            -> Pre computing even count
            EVEN_CT [VERTEX] += EVEN_CT [CHILD];
        }
}

INT MAIN ()
{
        INT n;
        CIN >> n;

        FOR (INT i=0; i < n-1; i++)
        {
            INT V1, V2;
            CIN >> V1 >> V2;
            G[V1].PB(V2);
            G[V2].PB(V1);
        }

        -> Running DFS first to precompute
           even_ct and subtree_sum
        DFS (1);

        -> Viewing subtree_sum and even_ct
           array:
        COUT << "SUBTREE SUM" << " " <<
        "EVEN COUNT" << "\n";
        FOR (INT i=1; i <= n; i++)
        {
```

```
        COUT << SUBTREE_SUM[i] << " "
        << EVEN_CT[i] << "\n";
    }

    RETURN 0;
}
```

INPUT:

```
13
1 2
1 3
1 13
2 5
3 4
3 6
5 7
5 8
8 12
4 9
4 10
10 11
```

OUTPUT:

```
SUBTREE SUM    EVEN COUNT
    91             6
    40             4
    37             2
    34             2
    38             3
     6             1
     7             0
    20             2
     9             0
    21             1
    11             0
    12             1
    13             0
```