PREREQUISITES:

Dynamic programming

PROBLEM:

Given a tree on N nodes where node i has value A_i, for each u from 1 to N compute $\sum_{i=1} \left\lfloor \frac{A_i}{2^{d(u,i)}} \right\rfloor$

EXPLANATION:

The main observation here is as follows: if $d(u, i) > 20$, then $\left\lfloor \frac{A_i}{2^{d(u,i)}} \right\rfloor = 0$ no matter what the value of A_i is, since $A_i \leq 10^6$.

This means that we only care about vertices at distances ≤ 20 from a given u.
Of course, this doesn't directly solve the problem, but it's a start.

Let's root the tree at some node, say 1.
With this root, let $p^i(u)$ denote the i-th ancestor of u. In particular, $p^0(u) = u$ and $p^1(u)$ is the parent of u.

Note that u contributes a value of $\left\lfloor \frac{A_u}{2^i} \right\rfloor$ to $p^i(u)$, and vice versa.

In particular, this allows us to, at least, compute the answer for every u when only considering values that lie in its subtree: for each u, add $\left\lfloor \frac{A_u}{2^i} \right\rfloor$ to the answer of $p^i(u)$ for each i from 0 to 20.
This takes $\mathcal{O}(20N)$ time.

Now, let's look at a specific u. We've already computed the contribution of things in its subtree, so we need to look outside.
So, let's look at $p^1(u)$. Consider some node v in the subtree of $p^1(u)$, that is not in the subtree of u.
If $d(v, p^1(u)) = k$, then $d(v, u) = k + 1$. Can we use this in some way?

Yes, we can!
Let's compute a 3D dynamic programming table: $dp[u][k][x]$ stores the following:

- Consider the subtree of vertex u, and all nodes at a distance of k from u in this subtree.
- $dp[u][k][x]$ holds the contribution of such nodes to a node at a distance of x from u.

So, coming back to our earlier discussion, the contribution of nodes in the subtree of $p^1(u)$ to the answer of u can be contributed using $dp[p^1(u)][k][1]$ across all k.
Note that this will also include some values in the subtree of u, which shouldn't be counted: their contribution can be subtracted out separately using the appropriate cell in the dp table.

Note that this allows us to visit every relevant ancestor of u and do the same thing. That is, for each $0 \leq i \leq 20$, visit $p^i(u)$ and add the values of $dp[p^i(u)][k][i]$ across all k, while also subtracting appropriate dp values to ensure that nothing is double-counted.
This will cover every node that is at a distance of ≤ 20 from u, which is exactly what we wanted.

The algorithm given above takes $\mathcal{O}(20^2 \cdot N)$ time and space.
It's possible to optimize the space to $\mathcal{O}(20N)$, but this optimization was unnecessary to get AC.