Colorful Trees

First, we can make the tree rooted for convenience.

Then, given an edge between a vertex and its parent like (v, p), we count the number of same-color pairs that use this edge for their connectivity.

Let's say cnt[c] is the total number of vertices with color c, and cnt[v][c] is the number of vertices in the subtree of v (all direct and indirect descendants of v, inclusive).

Clearly, the answer for the edge (v, p) will be:

Answer_(v, p) = Sum(cnt[v][c] * (cnt[c] - cnt[v][c])) for all possible colors

We observe that in cases where cnt[v][c] == 0 or cnt[v][c] == cnt[c], this term is equal to 0, so we can ignore these cases.

We can build cnt[v] by merging cnt[u] for all children u of v. These merges can be done using a DSU trick that results in O(n log n) operations on our collections. We can use either hash maps or tree maps for this.

Consider the cnt[v] for an imaginary vertex. As we are adding new vertices with any colors to this set, we can maintain the value of:

Sum(cnt[v][c] * (cnt[c] - cnt[v][c])) for all possible colors

in a variable like f[v]. In this way, we won't need to iterate over all existing colors in the set to calculate the final value for that vertex; instead, we can just use the value of f[v].

Here is a sample C++ code:

```cpp
void merge(int v, int u) { // merge u into v
    if (cnv[v].size() < cnv[u].size()) {
        cnv[v].swap(cnv[u]);
        swap(f[v], f[u]);
    }

    for (auto [x, cnx] : cnv[u]) {
        int av = cnv[v][x] * (cnt[x] - cnv[v][x]);
        f[v] -= av;

        cnv[v][x] += cnx;

        av = cnv[v][x] * (cnt[x] - cnv[v][x]);
        f[v] += av;

        if (av == 0) { // Not necessary
            cnv[v].erase(x);
        }
    }
}

void dfs(int v, int par = -1) {
    cnv[v][c[v]] = 1;
    f[v] = cnt[c[v]] - 1;

    for (auto [u, i] : g[v]) {
        if (u != par) {
            dfs(u, v);
            ans[i] = f[u];
            merge(v, u);
        }
    }
}
```