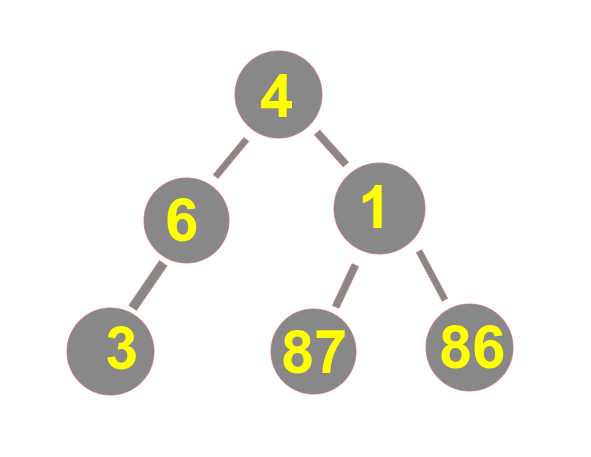
**XOR IS ZERO**

You are given binary tree, having integral (greater than 0) node values. Find the number of paths having bitwise XOR of all its node values 0.



For example, this binary tree has 2 paths(undirected) having Bitwise XOR as 0.

(3)-(6)-(4)-(1) and (87)-(1)-(86)

Note: Bitwise XOR is operation represented by ‘^’ on keyboard. Binary tree has nodes having at most 2 children.

**INPUT FORMAT:**

First line inputs **n**, number of nodes in Binary Tree.

Second line contains **n** integers, Array **A** representing node values.

Next **n-1** lines input undirected edges between diff nodes. It is guaranteed that the edges form a Binary Tree.

**OUTPUT FORMAT:**

Output single value, the number of paths having XOR of all values (of nodes) as 0.

**CONSTRAINTS:**

1 <= n <= 10^5

1 <= A[i] <= 127

**SAMPLE INPUT:**

6

4 6 1 3 87 86

0 1

1 3

0 2

2 4

2 5

**SAMPLE OUTPUT:**

As Explained above.

**SOLUTION:**

#include<bits/stdc++.h>

#define ll long long int

using namespace std;

ll n;

ll arr[100001];

vector<ll> adj[100001];

ll ans=0;

unordered\_map<ll,ll> findit(ll i, ll p)

{

//cout<<"i is:"<<i<<"\n";

int flg=0;

unordered\_map<ll,ll> mp1,mp2,mpans;

for(int j=0;j<adj[i].size();j++)

{

if(adj[i][j]!=p)

{

if(flg==0)

mp1=findit(adj[i][j], i);

else if(flg==1)

mp2=findit(adj[i][j], i);

flg++;

}

}

if(flg==0)

{

mpans[arr[i]]=1;

return mpans;

}

else if(flg==1)

{

ans+=mp1[arr[i]];

for(auto it = mp1.begin();it!=mp1.end();it++)

mpans[(it->first)^arr[i]]=it->second;

mpans[arr[i]]++;

return mpans;

}

else

{

for(auto it=mp1.begin();it!=mp1.end();it++)

{

ans+=(mp2[(it->first)^arr[i]]\*it->second);

}

ans+=mp1[arr[i]];

ans+=mp2[arr[i]];

for(auto it=mp1.begin();it!=mp1.end();it++)

{

mpans[it->first]+=it->second;

}

for(auto it=mp2.begin();it!=mp2.end();it++)

{

mpans[it->first]+=it->second;

}

mpans[arr[i]]++;

return mpans;

}

return mpans;

}

int main()

{

cin>>n;

for(int i=0;i<n;i++)

cin>>arr[i];

ll u,v;

for(int i=0;i<n-1;i++)

{

cin>>u>>v;

adj[u].push\_back(v);

adj[v].push\_back(u);

}

findit(0, -1);

cout<<ans<<"\n";

}