풀이

문제의 해는 두개의 특별한 정점 u, v에 대해 정점 u에서 정점 v로 가는 단순 경로에 속하는 모든 특별한 정점들을 하나의 집합으로 봤을때, 그러한 집합의 수가 최소가 되도록 나눴을 때의 집합의 수 -1개가 된다. (특별한 정점이 존재하지 않을 경우 0)

proof) 4개의 특별한 정점 u, v, x, y가 있고 u -> v, x -> y로 가는 겹치지 않는 단순 경로 집합이 있을때, 트리를 특별한 정점이 리프가 되도록 조작한 위상 동형인 트리로 생각하면 두 경로가 이어지기 위해서는 어떠한 리프노드 (u 또는 v)에서 다른 리프 노드 (x 또는 y)로 이동을 해야하는데, 그러기 위해서는 출발 리프노드의 부모 노드로 거슬러 올라가야한다. 하지만 문제의 특성상 모든 노드는 한번씩만 방문할 수 있고, u -> v (혹은 v -> u) 경로상에서 이미 양끝 리프 노드의 부모노드는 방문이 된 상태이므로 거슬러 올라갈 수 없고, 새롭게 추가된 간선 1개가 요구된다.

따라서 특별한 정점 집합의 최소 개수를 구하기 위해 다음을 정의한다.

**** dp[i] = 정점 i의 서브 트리에 해소되지 않은 특별한 정점 집합의 존재 여부 (존재시 1, 그렇지 않을 경우 0) ****

dp[i]의 전이는 아래와 같이

1. 기본적으로 정점 i가 특별한 노드일 경우 1을 가지게 된다,

2. 서브 트리를 조사하여 해소되지 않은 집합의 개수를 센다. 이 값들은 각각의 자식 노드의 dp값에 저장되게 된다.

3. 현재 정점의 서브트리에 존재하는 집합들은 현재 정점을 기준으로 나누어져 있고, 그들이 하나로 이어지기 위해서는 반드시 현재 정점을 통해 연결되어야하지만 현재 정점은 한번밖에 방문하지 못하므로 서브트리에 존재하는 두개의 집합만이 서로 연결될 수 있고 나머지 집합들은 연결하기 위해 집합당 1개의 추가 간선이 요구된다. 따라서 종합적으로 현재 노드의 서브 트리에 존재하는 해소되지 않은 집합이 k개라면 k-1개의 독립된 집합이 반드시 존재하게 되므로 이를 전체 집합의 개수에 누적한다.

4. 만약에 서브트리에 속하는 집합의 개수가 하나일 경우, 현재 노드 i의 조상 노드로 거슬러 올라가서 해소가 될 여지가 존재하기 때문에 현재 노드의 dp값을 1로 지정하여 전이한다.

5. 그렇지 않을 경우 현재 노드에서 그러한 집합들을 각각 카운트하여 해소하고 dp값을 0으로 지정하여 전이한다.

6. 루트 노드의 dp값이 최종적으로 1일 경우 따로 집합의 개수에 누적한다.

코드

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long
#define INF (int)1e18
#define endl "\n"

int res;
const int MAXN = 2e5+1;
vector<int> graph[MAXN], special, dp;

void dfs(int u, int parent) {
    dp[u] = special[u];
    int cnt = 0;
    for (auto v : graph[u]) {
        if (v == parent) continue;
        dfs(v, u);
        cnt += dp[v];
    }

    if (cnt > 0) res += (cnt-1);
    if (cnt == 1) dp[u] = 1;
    else if (cnt > 1) dp[u] = 0;
}

int32_t main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);

    int n; cin >> n;
    special.assign(n+1, 0); dp.assign(n+1, 0);
    for (int i = 0; i<n-1; i++) {
        int u, v; cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    for (int i = 1; i<=n; i++) cin >> special[i];

    res = 0;
    int root = 1;
    dfs(root, root);
    res += dp[root];

    cout << max(0LL, res - 1) << endl;

    return 0;
}
```