Day 97 coding Statement :

Arun has a rooted tree of N vertices rooted at vertex 1. Each vertex can either be coloured black or white.

Initially, the vertices are coloured A1?,A2?,…AN?, where Ai? ∈ {0,1} denotes the colour of the i-th vertex
(here 0 represents white and 1 represents black). He wants to perform some operations to change the colouring
of the vertices to B1?,B2?,…BN? respectively.

Arun can perform the following operation any number of times. In one operation, he can choose any subtree and
either paint all its vertices white or all its vertices black.

Help Arun find the minimum number of operations required to change the colouring of the vertices to B1?,B2?,…BN? respectively.

```
import java.util.*;
import java.lang.*;
import java.io.*;
class Main
{
public static void main (String[] args) throws java.lang.Exception
{
Scanner in = new Scanner(System.in);
int T = in.nextInt();
for (int t = 0; t < T; t++) {
 int N = in.nextInt();
 int[] arr1 = new int[N], arr2 = new int[N];
 for (int i = 0; i < N; i++) {
 arr1[i] = in.nextInt();
 }

 for (int i = 0; i < N; i++) {
 arr2[i] = in.nextInt();
 }
 List<Integer>[] adj = new ArrayList[N];
 for (int i = 0; i < N; i++) {
 adj[i] = new ArrayList<>();
 }
 for (int i = 0; i < N - 1; i++) {
 int u = in.nextInt() - 1, v = in.nextInt() - 1;
```

```
adj[u].add(v);
adj[v].add(u);
}
Queue<Integer> q = new LinkedList<>();
q.offer(0);
int[] depth = new int[N];
while (!q.isEmpty()) {
int current = q.poll();
for (int i : adj[current]) {
adj[i].remove(adj[i].indexOf(current));
depth[i] = depth[current] + 1;
q.offer(i);
}
}
int[] zero = new int[N], one = new int[N], none = new int[N];
List<Integer> sort = new ArrayList<>();
for (int i = 0; i < N; i++) {
sort.add(i);
}
Collections.sort(sort, (a, b) -> depth[b] - depth[a]);

for (int i : sort) {
int sumZero = 0, sumOne = 0, sumNone = 0;
for (int j : adj[i]) {
sumZero += zero[j];
sumOne += one[j];
sumNone += none[j];
}
if (arr2[i] == 0) {
zero[i] = sumZero;
one[i] = sumZero + 1;
none[i] = arr1[i] == 0 ? Math.min(sumNone, sumZero + 1) : sumZero + 1;
} else {
zero[i] = sumOne + 1;
one[i] = sumOne;
none[i] = arr1[i] == 1 ? Math.min(sumNone, sumOne + 1) : sumOne + 1;
}
}
System.out.println(none[0]);
}
}
}
```