

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 A_1, A_2, \dots, A_N , where $A_i \in \{0, 1\}$ 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 B_1, B_2, \dots, B_N 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 B_1, B_2, \dots, B_N respectively.

Input Format

- The first line contains a single integer T — the number of test cases. Then the test cases follow.
- The first line of each test case contains an integer N — the size of the tree.
- The second line of each test case contains N space-separated integers A_1, A_2, \dots, A_N denoting the initial colouring of the vertices of the tree.
- The third line of each test case contains N space-separated integers B_1, B_2, \dots, B_N denoting the final desired colouring of the vertices of the tree.
- The next $N-1$ lines contain two space-separated integers u and v — denoting an undirected edge between nodes u and v .

It is guaranteed that the edges given in the input form a tree.

Output Format

For each testcase, output the minimum number of operations required to obtain the desired colouring.

Sample Input

2

4

1 1 0 0

1 1 1 0

1 2

1 3

1 4

5

1 1 1 0 0

1 0 1 1 1

5 3

3 1

2 1

4 3

Sample Output

1

2

```
import java.util.*;
import java.lang.*;
import java.io.*;

class Program{
    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]);
}
}
}
}

```