```python
def dfs(vertex, parent, initial_colors, final_colors, adjacency_list, operations):
    if initial_colors[vertex] != final_colors[vertex]:
        operations[0] += 1
    for neighbor in adjacency_list[vertex]:
        if neighbor != parent:
            dfs(neighbor, vertex, initial_colors, final_colors, adjacency_list, operations)

# Read the number of test cases
T = int(input())

# Iterate over the test cases
for _ in range(T):
    # Read the size of the tree
    N = int(input())

    # Read the initial and final colorings of the vertices
    initial_colors = list(map(int, input().split()))
    final_colors = list(map(int, input().split()))

    # Create an adjacency list to represent the tree
    adjacency_list = [[] for _ in range(N+1)]

    # Read the N-1 edges of the tree
    for _ in range(N-1):
        u, v = map(int, input().split())
        adjacency_list[u].append(v)
        adjacency_list[v].append(u)

    # Initialize the operations count
    operations = [0]

    # Perform DFS traversal of the tree
    dfs(1, -1, initial_colors, final_colors, adjacency_list, operations)

    # Print the minimum number of operations required
    print(operations[0])
```