```java
package TREES;


import java.util.*;
public class count_path_in_vertical_direction_sum_modulo_k_is_zero {



    /*
    MEESHO | ONLINE ASSESSMENT | HASHING ON TREES
    Problem -
    A tree is given, with every node containing a value non-negative and a value m.
    We need to count the number of paths in the vertical direction such that
    the sum of nodes in the vertical direction modulo m is zero.

    Constrains:
    nodes <1e6
    nodes_val<1e9
    m<1e5

    Test Cases -
    4
    3 3
    1 2 3
    0 1
    1 2
    5 4
    2 1 3 4 2
    0 1
    0 2
    1 3
    1 4
    1 7
    5
    6 2
    2 1 2 1 2 2
    0 1
    0 2
    2 3
    1 4
    4 5
    6 3
    2 1 2 1 2 2
    0 1
    0 2
    2 3
    1 4
    4 5

    Ans-
    3
    1
    0
    6
    3

    Approach --
    Always when a tree problem is given think it of as a skewed linear tree or array
    Now , Think the same question in terms of array
    What is the number subarrays  till index i that meet the condition
    sum % m == 0

    Idea -- if[1...i] % m == 0 and [1.....j] % m == 0 where j>i
    then [i+1....j] == 0

    similarly if [1....i] % m == r and [1.......j] % m == r
    then also [i+1....j] % m ==0;
    we can say that person standing at i can look back that arr[i]%m was at how many places previously
    so from those many places we can form the subarrays.

    PsuedoCode
    map.put(0,1) //init now remainder is zero and found one time at the start
    ans = 0
    sum = 0
    for i = 1 to n:
        sum+=i
        rem = sum% m
        ans += map.get(rem);
        map.put(ans , freq++);
    return ans


    Now , if you are standing at index i you need and prefix sum [i] meaning sum till here.
    If the prefix sum till here is divisible % m == 0 then

    and if it is not divisible then ans[for this node] = 0

    At last take the sum of all answers.

    ans[i] -- stores the total number of subarrays that start from 1 and end at i whose sum is divisible by m







     */
    static int [] ans;
    static int [] val;
    static int running_sum;

    static HashMap<Integer , Integer > map;
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int t = s.nextInt();
        while(t-- >0) {
            int n = s.nextInt();
            int m = s.nextInt();
            List<List<Integer>> tree  = new ArrayList<>();
            val = new int[n];
            map = new HashMap<>();
            map.put(0, 1);
            boolean [] vis = new boolean[n];
            for( int i = 0 ; i <= n ; i++ ) tree.add(new ArrayList<>());
            for(int i = 0 ; i < n; i++ ) val[i] =  s.nextInt();
            for( int i = 0; i +1 < n ;i++ ){
                int u = s.nextInt() , v = s.nextInt();
                tree.get(u).add(v);
                tree.get(v).add(u);
            }
            dfs(0 , -1 , vis , tree ,m);

            int res = 0;
            for( int i: ans) res+=i;
            System.out.println(res);

        }

    }

    static void dfs(int node , int parent , boolean [] vis , List<List<Integer>>tree , 
int m) {
        vis[node]=true;
        running_sum+=val[node];
        int rem = (running_sum%m) < 0 ? running_sum%m +m : running_sum%m;
        int prev = map.getOrDefault(rem, 0);
        ans[node] += prev;
        map.put(rem , map.getOrDefault(rem,0)+1);

        // TRAVERSE
        for( int u : tree.get(node)) {
            if(u != parent && !vis[u]) {
                dfs(u , node , vis , tree,  m);
            }
        }

        running_sum -= val[node];
        int freq =  map.getOrDefault(rem,0);
         if(freq == 0) {
            map.remove(rem);
         }
         else map.put(rem , freq-1);
    }
}
```