Problem 1: Zero Sum Tree Partition

Problem Statement

You are given a tree with N nodes and N-1 edges.

Each node has a value either 1 or -1, given in an array val of size N.

You are allowed to remove any subset of edges (including none).

Your task is to count the number of distinct edge subsets such that after removing those edges,

the resulting connected components each have a total node value of 0.

Return the count of such valid edge subsets modulo 10 + 7.

Input:

- int N: number of nodes

- List<Integer> val: node values (1-based)

- List<List<Integer>> edges: N-1 edges, where each edge is a list of two integers [u, v] (1-based index)

Constraints:

- 1 N 10

- val[i] {1, -1}

- 1 u, v N

Output:

- Integer: Number of valid edge subsets mod 1e9 + 7

Example:

Input:

N = 4

val = [1, -1, 1, -1]

edges = [[1, 2], [2, 3], [3, 4]]

Output:

2

Explanation:

Two valid edge subsets:

- No edge removed (entire tree sum = 0)

- Remove edge between node 2 and 3 Two components with [1,2] = 0 and [3,4] = 0

Java Solution

```
import java.util.*;

public class ZeroSumTreePartition {
    static final int MOD = 1_000_000_007;
    static List<List<Integer>> tree;
    static int[] val;
    static int zeroSumCuts;

    public static int countEdgeSubsets(int n, List<Integer> valList, List<List<Integer>>
edges) {
        val = new int[n + 1];
        for (int i = 0; i < n; i++) val[i + 1] = valList.get(i);

        tree = new ArrayList<>();
        for (int i = 0; i <= n; i++) tree.add(new ArrayList<>());
        for (List<Integer> edge : edges) {
            int u = edge.get(0), v = edge.get(1);
            tree.get(u).add(v);
            tree.get(v).add(u);
        }

        int total = 0;
        for (int i = 1; i <= n; i++) total += val[i];
        if (total != 0) return 0;

        zeroSumCuts = 0;
        dfs(1, -1);
        return (int) modPow(2, zeroSumCuts, MOD);
    }

    private static int dfs(int node, int parent) {
        int sum = val[node];
        for (int child : tree.get(node)) {
            if (child != parent) {
                int childSum = dfs(child, node);
                if (childSum == 0) zeroSumCuts++;
                else sum += childSum;
            }
        }
        return sum;
    }

    private static long modPow(long base, int exp, int mod) {
        long res = 1;
        while (exp > 0) {
            if ((exp & 1) == 1) res = (res * base) % mod;
            base = (base * base) % mod;
            exp >>= 1;
        }
        return res;
    }
}
```