Problem of the Week(Hard)

Company: Adobe
Topic: Trees, DFS

Scenario

You are working on a problem where a tree (undirected, connected graph with `N` nodes and `N-1` edges) is given. The tree has an **even number of nodes**. Your task is to remove as many edges as possible such that every connected component (subtree) left after removals contains an even number of nodes.

This problem is directly related to **Tree DFS, Subtree Sizes, and Graph Partitioning**.

Problem Statement

You are given a tree with `N` nodes, where `N` is even. Each edge connects a parent node with a child node. You need to remove the **maximum number of edges** such that all resulting disconnected subtrees have an **even number of nodes**.

Input Format

- First line: An integer `N` (number of nodes, even).
- Next `N-1` lines: Two integers `u v` denoting an edge between node `u` and node `v`.

Output Format

- Print the maximum number of edges that can be removed while still ensuring that every resulting subtree has an even number of nodes.

Example

Input:

```
8
1 2
1 3
```

```
3 4
3 5
4 6
4 7
4 8
```

Output:

```
2
```

Explanation:

- Subtree sizes:
  - Removing edge `(3,4)` → subtree with nodes `{4,6,7,8}` (size 4, even).
  - Removing edge `(1,3)` → subtree `{3,5}` (size 2, even).
- Both removals are valid, so answer is `2`.

Approach

Approach 1: DFS with Subtree Sizes (Efficient)

1. Perform a **DFS** from the root (say node 1).
2. Calculate the size of each subtree.
3. If a subtree rooted at `v` has an **even size**, then the edge connecting it to its parent can be removed.
4. Count such removable edges.

- **Time Complexity:** `O(N)`
- **Space Complexity:** `O(N)`

Approach 2: Brute Force (Not Recommended)

1. Try removing edges one by one and check if all resulting subtrees have even sizes.
2. Too costly → `O(N^2)` in worst case.

Practice Links

- GeeksforGeeks – Even Forest Problem
- HackerRank – Even Tree

Reference Code (Java)

```java
import java.util.*;

public class EvenTree {
    static List<List<Integer>> adj;
    static int removableEdges = 0;

    public static int dfs(int node, int parent) {
        int subtreeSize = 1;
        for (int child : adj.get(node)) {
            if (child != parent) {
                int childSize = dfs(child, node);
                if (childSize % 2 == 0) {
                    removableEdges++;
                } else {
                    subtreeSize += childSize;
                }
            }
        }
        return subtreeSize;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        adj = new ArrayList<>();
        for (int i = 0; i <= n; i++) adj.add(new ArrayList<>());

        for (int i = 0; i < n - 1; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            adj.get(u).add(v);
            adj.get(v).add(u);
        }

        dfs(1, -1);
        System.out.println(removableEdges);
    }
}
```