```
class Solution {
    public int countComponents(int n, int[][] edges) {
    int[] roots = new int[n];
    for(int i = 0; i < n; i++) roots[i] = i;

    for(int[] e : edges) {
        int root1 = find(roots, e[0]);
        int root2 = find(roots, e[1]);
        if(root1 != root2) {
            roots[root1] = root2;  // union
            n--;
        }
    }
    return n;
}

    private int find(int[] nodes, int node) {
                    if (nodes[node] == node) return node;
                    nodes[node] = find(nodes, nodes[node]);
                    return nodes[node];
            }
/*
public int find(int[] roots, int id) {
    while(roots[id] != id) {
        roots[id] = roots[roots[id]];  // optional: path compression
        id = roots[id];
    }
    return id;
}
*/
    /*
    public int countComponents(int n, int[][] edges) {
        List<List<Integer>> adj = buildAdj(edges, n);
        int cnt = 0;
        boolean[] connected = new boolean[n];

        for (int i = 0; i < connected.length; i++) {
            if (!connected[i]) {
                cnt++;
                connect(adj, i, connected);
            }
        }
        return cnt;
    }

    private void connect(List<List<Integer>> adj, int idx, boolean[] connected) {
        if (connected[idx])
            return;
        connected[idx] = true;
        for (int i : adj.get(idx)) {
            connect(adj, i, connected);
        }
    }
    private List<List<Integer>> buildAdj(int[][] edges, int n) {
        List<List<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            adj.add(new ArrayList<>());
        }

        for (int[] e : edges) {
            adj.get(e[0]).add(e[1]);
            adj.get(e[1]).add(e[0]);
        }

        return adj;
    }
    */
}
```