2872. Maximum Number of K-Divisible Components

Solved

Hard | Topics | Companies | Hint

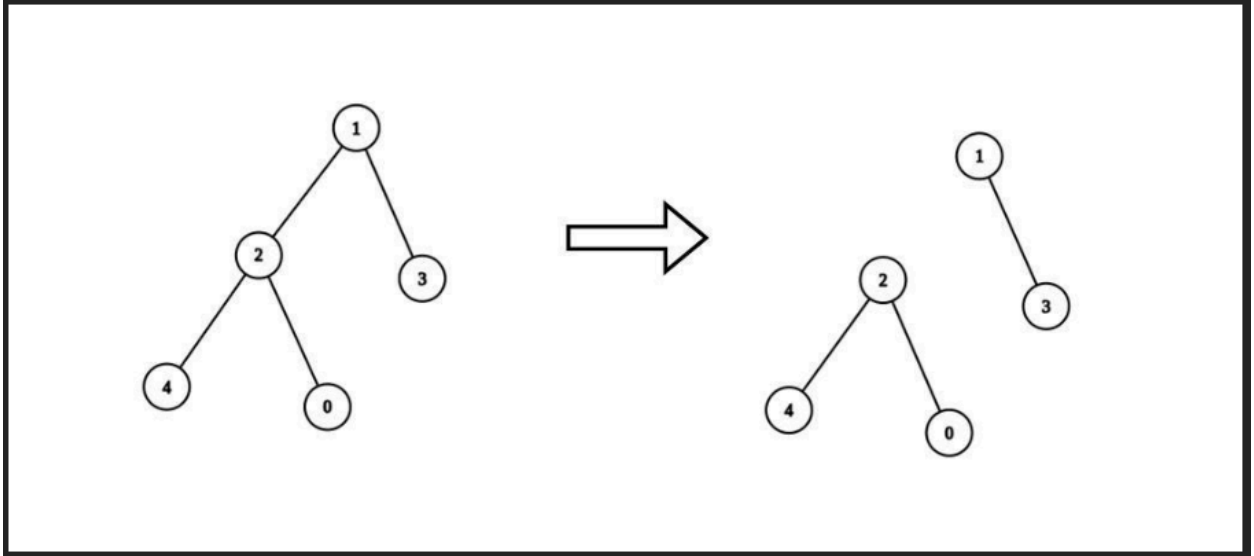
There is an undirected tree with `n` nodes labeled from `0` to `n - 1`. You are given the integer `n` and a 2D integer array `edges` of length `n - 1`, where `edges[i] = [a_i, b_i]` indicates that there is an edge between nodes a_i and b_i in the tree.

You are also given a **0-indexed** integer array `values` of length `n`, where `values[i]` is the **value** associated with the i^{th} node, and an integer `k`.

A **valid split** of the tree is obtained by removing any set of edges, possibly empty, from the tree such that the resulting components all have values that are divisible by `k`, where the **value of a connected component** is the sum of the values of its nodes.

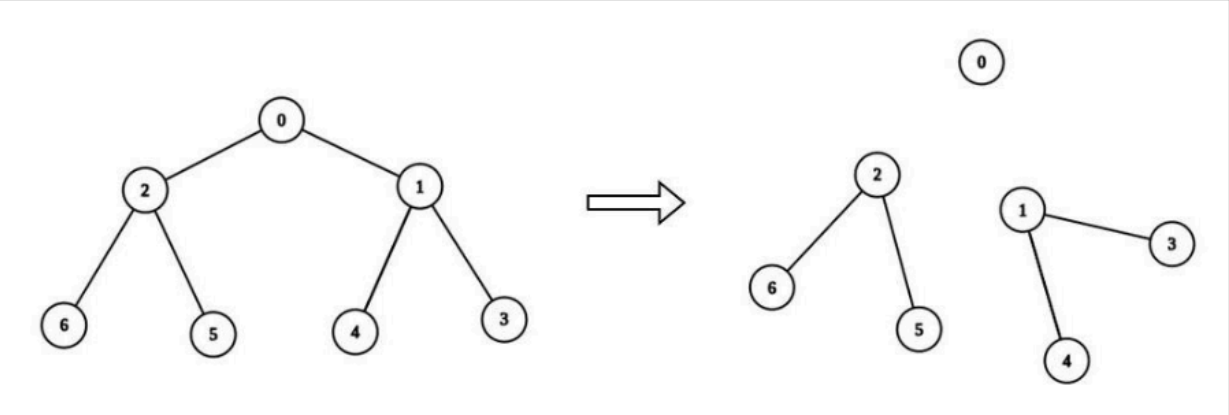
Return *the **maximum number of components** in any valid split.*

Example 1:




```
Input: n = 5, edges = [[0,2],[1,2],[1,3],[2,4]], values = [1,8,1,4,4], k = 6
Output: 2
Explanation: We remove the edge connecting node 1 with 2. The resulting split
is valid because:
- The value of the component containing nodes 1 and 3 is values[1] + values[3]
= 12.
- The value of the component containing nodes 0, 2, and 4 is values[0] +
values[2] + values[4] = 6.
It can be shown that no other valid split has more than 2 connected components.
```

Example 2:




```
Input: n = 7, edges = [[0,1],[0,2],[1,3],[1,4],[2,5],[2,6]], values =
[3,0,6,1,5,2,1], k = 3
Output: 3
Explanation: We remove the edge connecting node 0 with 2, and the edge
connecting node 0 with 1. The resulting split is valid because:
- The value of the component containing node 0 is values[0] = 3.
- The value of the component containing nodes 2, 5, and 6 is values[2] +
values[5] + values[6] = 9.
- The value of the component containing nodes 1, 3, and 4 is values[1] +
values[3] + values[4] = 6.
It can be shown that no other valid split has more than 3 connected components.
```

Constraints:

- $1 <= n <= 3 * 10^4$
- `edges.length == n - 1`
- `edges[i].length == 2`
- $0 <= a_i, b_i < n$
- `values.length == n`
- $0 <= values[i] <= 10^9$
- $1 <= k <= 10^9$
- Sum of `values` is divisible by `k`.
- The input is generated such that `edges` represents a valid tree.

Python:

```python
class Solution:
    def maxKDivisibleComponents(self, n: int, edges: List[List[int]], values: List[int], k: int) -> int:
        adj = defaultdict(list)
        src = 0
        for u,v in edges:
            adj[u].append(v)
            adj[v].append(u)
        comp = 0
        visited = set()
        def dfs(root):
            nonlocal comp
            if root in visited:
                return 0
            visited.add(root)
            ans = values[root]
            for neigh in adj[root]:
                ans += dfs(neigh)
            if ans % k == 0:
                comp += 1
                return 0
            return ans % k
        dfs(src)
        return comp
```

JavaScript:

```javascript
/**
 * @param {number} n
 * @param {number[][]} edges
 * @param {number[]} values
 * @param {number} k
 * @return {number}
 */
var maxKDivisibleComponents = function(n, edges, values, k) {
    // Step 1: Build the adjacency list representation of the tree
    const tree = Array.from({ length: n }, () => []);
    for (const [a, b] of edges) {
        tree[a].push(b);
        tree[b].push(a);
    }

    // Step 2: Perform DFS to calculate subtree sums
    const subtreeSum = Array(n).fill(0);
    const visited = Array(n).fill(false);
```

```
    function dfs(node) {
        visited[node] = true;
        let sum = values[node];

        for (const neighbor of tree[node]) {
            if (!visited[neighbor]) {
                sum += dfs(neighbor);
            }
        }

        subtreeSum[node] = sum;
        return sum;
    }

    // Perform DFS from the root (node 0)
    dfs(0);

    // Step 3: Check for valid splits
    let maxComponents = 1; // At least one component (the whole tree)

    function dfsCount(node) {
        visited[node] = true;
        let componentCount = 0;

        for (const neighbor of tree[node]) {
            if (!visited[neighbor]) {
                if (subtreeSum[neighbor] % k === 0) {
                    componentCount++; // Split here
                }
                componentCount += dfsCount(neighbor);
            }
        }

        return componentCount;
    }

    // Reset visited array and count components
    visited.fill(false);
    maxComponents += dfsCount(0);

    return maxComponents;
};
```

Java:

```
class Solution {
    private Map<Integer, List<Integer>> adj;
    private Set<Integer> visited;
    private int comp;

    public int maxKDivisibleComponents(int n, int[][] edges, int[] values, int k) {
        adj = new HashMap<>();
        visited = new HashSet<>();
        comp = 0;

        int src = 0;

        for (int[] edge : edges) {
            int u = edge[0];
            int v = edge[1];
            adj.computeIfAbsent(u, k1 -> new ArrayList<>()).add(v);
            adj.computeIfAbsent(v, k1 -> new ArrayList<>()).add(u);
        }

        dfs(src, values, k);
        return comp;
    }

    private int dfs(int root, int[] values, int k) {
        if (visited.contains(root)) {
            return 0;
        }

        visited.add(root);
        int ans = values[root];

        for (int neigh : adj.getOrDefault(root, Collections.emptyList())) {
            ans += dfs(neigh, values, k);
        }

        if (ans % k == 0) {
            comp++;
            return 0;
        }

        return ans % k;
    }}
```