**[3425. Longest Special Path](https://leetcode.com/problems/longest-special-path/)**

You are given an undirected tree rooted at node 0 with n nodes numbered from 0 to n - 1, represented by a 2D array edges of length n - 1, where edges[i] = [ui, vi, lengthi] indicates an edge between nodes ui and vi with length lengthi. You are also given an integer array nums, where nums[i] represents the value at node i.

A **special path** is defined as a **downward** path from an ancestor node to a descendant node such that all the values of the nodes in that path are **unique**.

**Note** that a path may start and end at the same node.

Return an array result of size 2, where result[0] is the **length** of the **longest** special path, and result[1] is the **minimum** number of nodes in all *possible* **longest** special paths.

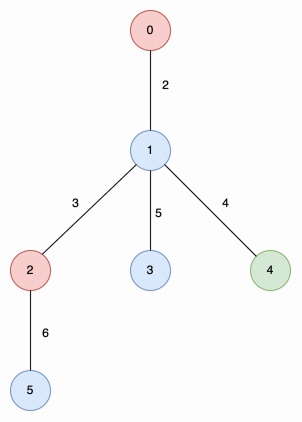
**Example 1:**

**Input:** edges = [[0,1,2],[1,2,3],[1,3,5],[1,4,4],[2,5,6]], nums = [2,1,2,1,3,1]

**Output:** [6,2]

**Explanation:**

**In the image below, nodes are colored by their corresponding values in nums**



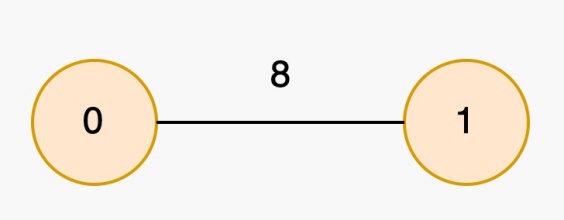
The longest special paths are 2 -> 5 and 0 -> 1 -> 4, both having a length of 6. The minimum number of nodes across all longest special paths is 2.

**Example 2:**

**Input:** edges = [[1,0,8]], nums = [2,2]

**Output:** [0,1]

**Explanation:**



The longest special paths are 0 and 1, both having a length of 0. The minimum number of nodes across all longest special paths is 1.

**Constraints:**

* 2 <= n <= 5 \* 104
* edges.length == n - 1
* edges[i].length == 3
* 0 <= ui, vi < n
* 1 <= lengthi <= 103
* nums.length == n
* 0 <= nums[i] <= 5 \* 104
* The input is generated such that edges represents a valid tree.

Solution

import sys

sys.setrecursionlimit(10\*\*7)

class Solution:

    def longestSpecialPath(self, edges, nums):

        n = len(nums)

        adj = [[] for \_ in range(n)]

        for u,v,w in edges:

            adj[u].append((v,w))

            adj[v].append((u,w))

        dist = [0]\*n

        startIndex = [0]\*n

        lastPos = {}

        stack = []

        bestLen = 0

        bestCount = 1

        def dfs(u, p):

            nonlocal bestLen, bestCount

            stack.append(u)

            idx = len(stack) - 1

            old = lastPos.get(nums[u], -1)

            lastPos[nums[u]] = idx

            if p == -1:

                startIndex[u] = 0

            else:

                startIndex[u] = max(startIndex[p], old + 1 if old != -1 else 0)

            if startIndex[u] <= idx:

                L = dist[u] - dist[stack[startIndex[u]]]

                cnt = idx - startIndex[u] + 1

            else:

                L = 0

                cnt = 1

            if L > bestLen:

                bestLen = L

                bestCount = cnt

            elif L == bestLen and cnt < bestCount:

                bestCount = cnt

            for v, w in adj[u]:

                if v != p:

                    dist[v] = dist[u] + w

                    dfs(v, u)

            lastPos[nums[u]] = old

            stack.pop()

        dfs(0, -1)

        return [bestLen, bestCount]