🔴🔥 HARD QUESTIONS……..

1 — Tree Color Flip Minimum Operations

(TREE + DFS + FLIP PARITY DP)

📝 **Problem**

You have a tree with n nodes.
Each node has:

- Initial color: A[i] (0 or 1)
- Target color: B[i] (0 or 1)

You can perform 2 types of operations:

🟢 **1. Flip a single node**

🔵 **2. Flip the entire subtree rooted at that node**

Flip means:

0 ↔ 1

Goal:
➡️ Transform all A[i] → B[i]
➡️ Using **minimum number of operations**

📥 **Input**

n

A[1..n]

B[1..n]

edges...

🎯 **Constraints**

1 ≤ n ≤ 200000

⭐ **Key Concept**

When you flip a subtree, every descendant changes → propagate a flip parity bit.

⭐ **Code**

```
import sys

sys.setrecursionlimit(300000)


n = int(input())

A = [0] + list(map(int, input().split()))

B = [0] + list(map(int, input().split()))


g = [[] for _ in range(n+1)]

for _ in range(n-1):

    u,v = map(int, input().split())

    g[u].append(v)

    g[v].append(u)


ans = 0


def dfs(u, p, flip):

    global ans

    cur = A[u] ^ flip


    if cur != B[u]:

        ans += 1

        flip ^= 1


    for v in g[u]:

        if v != p:
```

```
        dfs(v, u, flip)


dfs(1, -1, 0)
print(ans)
```

🔴🔥 HARD QUESTION 2 — Graph Binary Coloring with Constraints

(BIPARTITE CHECK + PRE-COLORING + MINIMIZATION)

📝 Problem

An undirected graph with n nodes.

Each node has:

color[i] = -1 (uncolored)

= 0 (fixed color)

= 1 (fixed color)

You must color all uncolored nodes (0 or 1) such that:

✓ Graph becomes bipartite

✓ No conflict with preset colors

✓ Minimize total nodes colored as 1

If impossible → print -1.

📥 Input

n m

color[1..n]

edges...

⭐ Code

```
import sys
```

```
sys.setrecursionlimit(300000)

n,m = map(int, input().split())
col = [0] + list(map(int, input().split()))

g = [[] for _ in range(n+1)]
for _ in range(m):
    u,v = map(int, input().split())
    g[u].append(v)
    g[v].append(u)

vis = [False]*(n+1)
ans = 0

def dfs(u, c0, comp):
    vis[u] = True
    comp.append(u)

    if col[u] != -1 and col[u] != c0:
        return False

    if col[u] == -1:
        col[u] = c0

    for v in g[u]:
        if not vis[v]:
            if not dfs(v, 1-c0, comp):
                return False
```

```
        elif col[v] == col[u]:
            return False

    return True

ok = True
for i in range(1, n+1):
    if not vis[i]:
        comp = []
        if not dfs(i, 0, comp):
            ok = False
            break
        ans += sum(col[x] for x in comp)

print(ans if ok else -1)
```

🔴🔥 **HARD QUESTION 3 — XOR Non-Decreasing Subsequence**

(DP with XOR state 0–1023)

📝 **Problem**

Given array A of size n.
Find the **longest NON-DECREASING subsequence** such that:

xor of chosen elements >= M

📥 **Input**

n M

A[1..n]

🔒 **Constraints**

A[i] < 1024

⭐ **Code**

```
n, M = map(int, input().split())
A = list(map(int, input().split()))

dp = [[0]*1024 for _ in range(n)]
ans = 0

for i in range(n):
    dp[i][A[i]] = 1
    for j in range(i):
        if A[j] <= A[i]:
            for x in range(1024):
                if dp[j][x]:
                    dp[i][x^A[i]] = max(dp[i][x^A[i]], dp[j][x] + 1)
    for x in range(1024):
        if x >= M:
            ans = max(ans, dp[i][x])

print(ans)
```

🔴🔥 **HARD QUESTION 4 — Tree Value Compression + Pair Counting**

(Tree DP + Hashmap + Square-Free)

📝 **Problem**

Given tree with values a[i].

Count total number of pairs (u,v) in whole tree such that:

gcd(a[u], a[v]) = 1

Return count.

🔧 **Key Concept**

This is **global pair count using tree DP**, similar to beauty problem but gcd-based.

⭐ **Code**

```
from collections import defaultdict
import sys
from math import gcd
sys.setrecursionlimit(300000)

n = int(input())
a = [0] + list(map(int, input().split()))

g = [[] for _ in range(n+1)]
for _ in range(n-1):
    u,v = map(int, input().split())
    g[u].append(v)
    g[v].append(u)

ans = 0

def dfs(u,p):
    global ans
    mp = defaultdict(int)
    mp[a[u]] = 1
```

```
    for v in g[u]:
        if v==p: continue
        child = dfs(v,u)
        if len(child) > len(mp):
            mp, child = child, mp
        for x in child:
            for y in mp:
                if gcd(x,y) == 1:
                    ans += child[x] * mp[y]
        for x in child:
            mp[x] += child[x]

    return mp

dfs(1,-1)
print(ans)
```

🔴🔥 **HARD QUESTION 5 — DAG XOR Path Constraint**

(Graph DP on Topo Order + XOR DP)

📝 **Problem**

Directed Acyclic Graph (DAG).
Each edge has weight w.

A path is **valid** iff:

(sum of weights along path) XOR (number of nodes in path) <= K

Find **longest valid path**.

📥 **Input**

n m K

edges: u v w

⭐ **Code**

```
from collections import deque, defaultdict


n,m,K = map(int, input().split())
g = defaultdict(list)
ind = [0]*(n+1)


for _ in range(m):
    u,v,w = map(int, input().split())
    g[u].append((v,w))
    ind[v] += 1


q = deque()
for i in range(1,n+1):
    if ind[i] == 0:
        q.append(i)


top = []
while q:
    u = q.popleft()
    top.append(u)
    for v,_ in g[u]:
        ind[v] -= 1
        if ind[v] == 0:
```

```
            q.append(v)


dp = [[-1]*1024 for _ in range(n+1)]
ans = 0


for u in top:
    if dp[u][0] == -1:
        dp[u][0] = 1


    for x in range(1024):
        if dp[u][x] == -1:
            continue
        for v,w in g[u]:
            nx = x ^ w
            if nx <= K:
                dp[v][nx] = max(dp[v][nx], dp[u][x] + 1)


    ans = max(ans, max(dp[u]))


print(ans)
```

🎉 **HARD SECTION COMPLETE**