TEAMFULNESS

Statement: You are given a tree of N vertices. Each vertex is coloured one of K colours. Your task is to count the sum of the number of distinct colours for all paths in the tree that are of a maximum possible length.

Constraints:

N <= 10^6

K <= N

TEAMFULNESS

Definition: the diameter of a tree is the length of its longest simple path. Any path achieving this length is called a diameter.
Fact: every diameter's endpoints are leaves — if an endpoint had an unused neighbour, we could walk onto it and get something longer than D.

TEAMFULNESS

Lemma (double-BFS / double-DFS):

- Start at any node r; run BFS/DFS to find the farthest node from r — call it A.
- Run BFS/DFS from A to find the farthest node from A — call it B.
- The path A–B is guaranteed to be a diameter, and D = dist(A, B).

This works because of a short sub-lemma: for any starting node r, the farthest node from r is always an endpoint of some diameter. That's why fixing A first and re-searching from A is guaranteed to land on the true other end.

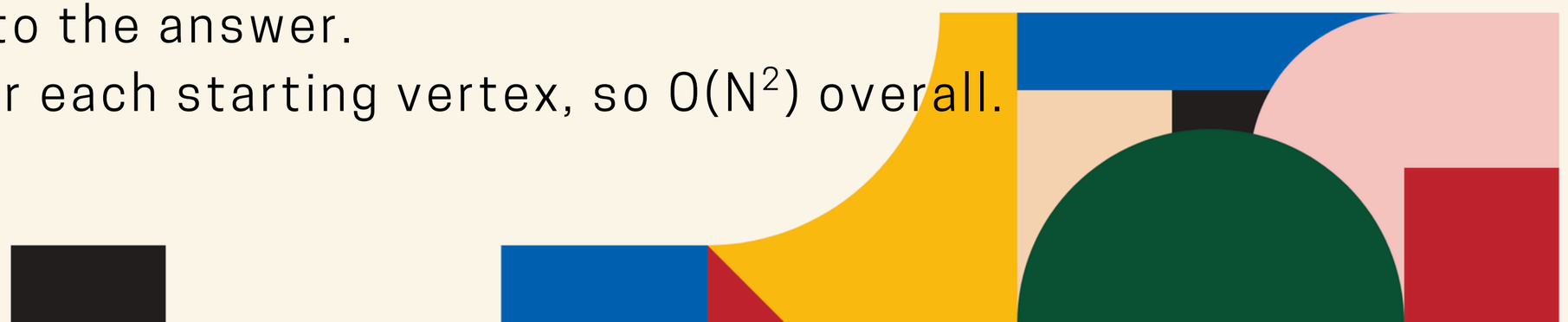
TEAMFULNESS

O(N^2) solution

First, find D (diameter's length) using the described algorithm. Then, for every starting vertex:

- run DFS,
- maintain how many times every colour appears on the current path,
- also maintain the number of distinct colours currently present.

Whenever we reach a leaf whose depth is exactly D,
we have found one possible diameter endpoint.
The current path has length D, so we simply add the number of distinct colours on this path to the answer.
Complexity is O(N) for each starting vertex, so O(N^2) overall.



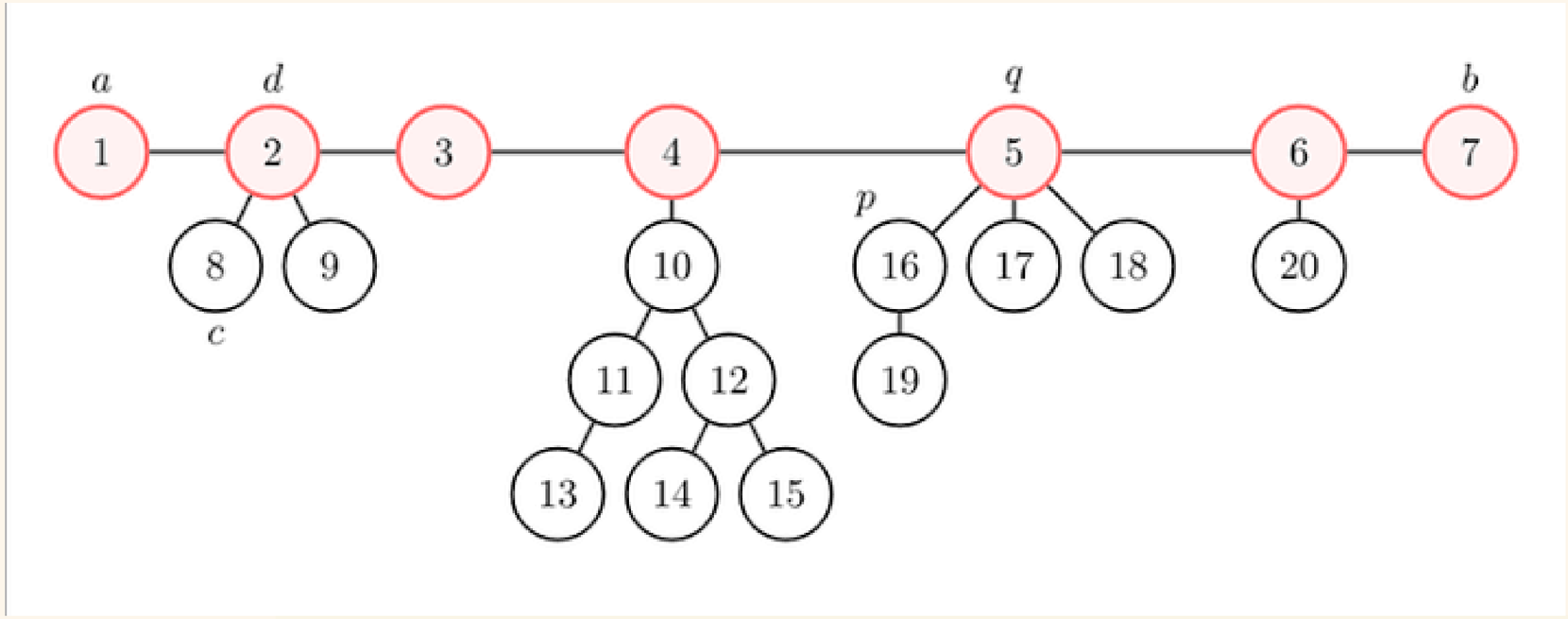
TEAMFULNESS

The centre of the tree

Every diameter passes through the ***centre***, split into two equal halves:

- **D even:** every diameter = (path from centre to leaf X) + (path from centre to leaf Y), where X and Y sit in two different subtrees hanging off the centre.
- **D odd:** every diameter = (path from u into u's side) + (the center edge u–v) + (path from v into v's side).

TEAMFULNESS



TEAMFULNESS

Now fix this center.
Removing the center splits the tree into **several connected components.**
Every diameter can be viewed as two paths starting from the center, each ending at a leaf in a different component.
However, not every leaf can be an endpoint of a diameter. A leaf is an **endpoint** of some diameter if and only if it is at distance D/2 from the center (or from the center edge in the odd case), where D is the diameter length.
We'll call such leaves **valid endpoints.**
So from now on, instead of thinking about diameters, we only need to think about **pairs of valid endpoints lying in different components.**

TEAMFULNESS

Subtask 5: k = 1.

When K = 1, every diameter contributes exactly 1. So we only need to count how many diameters exist. Suppose that for every component around the centre, we know: **cnt[i] = the number of valid endpoints in this component.**

So the answer is simply the sum of

cnt[i] × cnt[j] over all pairs of different components. Calculating this could be easily optimised by maintaining some variable **sum_cnt** (the total sum of cnt[i] over processed components).

Then we add **sum_cnt × cnt[j]** to the answer every time and update sum_cnt after that.

TEAMFULNESS

Let's reformulate the problem.
Instead of counting diameters directly, let's count, **for each colour** independently, **how many diameters contain that colour.**
The final answer is simply the sum of these values over all colours.
Fix one colour and mark every vertex of this colour.
Now, for every component around the center, compute two values:

- $good_i$ — the number of valid endpoints whose path to the center contains at least one marked vertex.
- bad_i — the number of valid endpoints whose path to the center contains no marked vertices.

TEAMFULNESS

How do we compute $good_i$ and bad_i?

Run a DFS/BFS from the center.

Maintain a variable vis, equal to the number of marked vertices on the current path from the center to the current node.

- When entering a marked vertex, increment vis.
- When leaving it, decrement vis.

Whenever we reach a valid endpoint:

- if vis > 0, this endpoint contributes to $good_i$,
- otherwise, it contributes to bad_i.

Thus, one traversal is enough to compute $good_i$ and bad_i for every component.

TEAMFULNESS

Counting the diameters

Now consider two components, i and j.

A diameter contains the chosen colour unless both endpoint-to-center paths avoid it.

There are therefore three valid cases:

- both endpoints are good,
- the endpoint in component i is good and the one in j is bad,
- the endpoint in component i is bad and the one in j is good.

Hence the number of diameters using these two components is

$$\mathbf{good_i \times good_j + good_i \times bad_j + bad_i \times good_j.}$$

TEAMFULNESS

Iterating over all pairs of components would be quadratic.
Instead, we process the components one by one while maintaining:

- **sum_good** — the total number of good endpoints in all previously processed components,
- **sum_bad** — the total number of bad endpoints in all previously processed components.

When processing a new component i, its contribution is
$\text{sum\_good} \times \text{bad}_i + \text{sum\_bad} \times \text{good}_i + \text{sum\_good} \times \text{good}_i$.
Each component is processed in O(1), so for one colour the complexity is O(N), and over all colours it becomes $O(N \cdot K)$.

TEAMFULNESS

Full solution

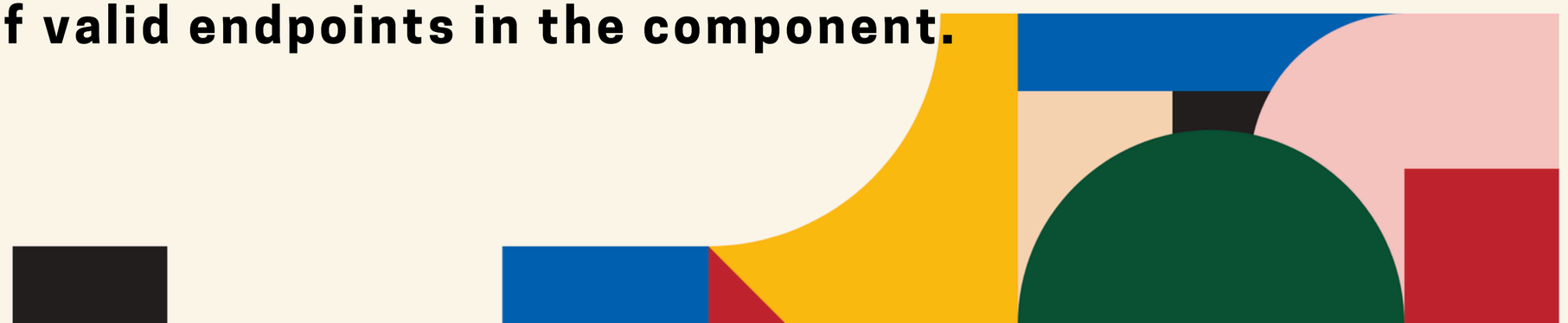
The main idea remains exactly the same—we still perform a DFS and maintain the vis counters—but now we process all colours **simultaneously**.

Maintain vis[c] for every colour c, where vis[c] is the number of vertices of colour c on the current DFS path.

Suppose we enter a vertex v of colour c. If vis[c] was 0 before entering v, then v is the first occurrence of colour c on the current path. Consequently, every valid endpoint inside the subtree of v must have colour c on its path to the center.

TEAMFULNESS

To exploit this observation, precompute
ends[v] = number of valid endpoints in the subtree of v.
Then, whenever vis[c] changes from 0 to 1, we immediately know that all these endpoints become good for colour c, so we simply do
good$_i$[c] += ends[v].
This counts all those endpoints at once, instead of processing them individually.
We cannot compute bad$_i$[c] for every colour efficiently.
Instead, for each component we only store
all$_i$ = total number of valid endpoints in the component.



TEAMFULNESS

Assume we have already computed $good_i[c]$ for every component and every colour appearing in it.
First, let's process the components from left to right.
We maintain **sum_all** = the total number of valid endpoints in the processed components.
When processing component i, for every colour c present in it, add **$sum\_all \times good_i[c]$** to the answer.
This counts all pairs where the endpoint in the current component is good, namely:

- **bad × good,**
- **good × good.**

TEAMFULNESS

Now process the components in reverse order and maintain:

- **sum_all,**
- **sum_good[c] = total value of good[c] over all already processed components.**

For every colour c, add **$good_i[c] \times (sum\_all - sum\_good[c])$.**
The term **sum_all − sum_good[c]** is exactly the number of endpoints that are bad for colour c in the remaining components, so this counts **the missing good × bad pairs.**
Together, the two passes count every valid pair exactly once, giving the final linear-time solution.