Editorial: Component Colors

1 Core Observation

The key insight for solving the problem is recognizing that the connected components formed by edges with weight at most a threshold w can be efficiently represented and queried using a Kruskal Reconstruction Tree (minimized for threshold queries). By processing edges in increasing order and constructing a tree where each internal node corresponds to a component merge (annotated with the merging edge's weight), we create a structure where the connected component for any threshold w is represented by the highest ancestor node (in this tree) with weight $\leq w$. Additionally, storing distinct color counts per component via small-to-large set merging during tree construction enables efficient query resolution.

2 Solution

We solve the problem through the following steps:

Preprocessing

1. **Build Kruskal Reconstruction Tree (min tree):**
   - Sort all m edges by weight in increasing order.
   - Initialize a DSU to track connected components. Each component stores:
     - Root node in the reconstruction tree.
     - A set of distinct colors (initially the node's color) and the distinct color count.
   - For each edge (u, v) in sorted order (weight w):
     - Find DSU representatives for u and v. Skip if they are the same.
     - Create a new internal tree node x with weight w.
     - Set x as parent of the DSU roots for u and v's components.
     - Merge the color sets from both components into x using small-to-large:
       * Always merge the smaller set into the larger set.
       * Update the distinct color count for x as the size of the merged set.
     - Update the DSU: merge components and set x as the new root.
2. **Binary Lifting Setup:**
   - The reconstruction tree has $N = 2n - 1$ nodes (original nodes + internal nodes).
   - For each node, precompute binary lifting table $\text{up}[i][j]$ (ancestor 2^j levels up).
   - Initialize: $\text{up}[i][0] = \text{parent}(i)$ for all nodes (leaves have parent $= -1$).
   - For $j = 1$ to $\text{MAX\_LOG} - 1$:

$$\text{up}[i][j] = \begin{cases} \text{up}[\text{up}[i][j-1]][j-1] & \text{if up}[i][j-1] \neq -1 \\ -1 & \text{otherwise} \end{cases}$$

Query Processing

For each query (v, w):

1. Start at leaf node v.
2. Traverse up using binary lifting:

$u \leftarrow v$
for $j = \text{MAX\_LOG} - 1$ **downto** 0 **do**
  if $\text{up}[u][j] \neq -1$ **and** $\text{weight}[\text{up}[u][j]] \leq w$ **then**
    $u \leftarrow \text{up}[u][j]$
  end if
end for
return $\text{distinct}[u]$ {Distinct color count at node u}

3 Complexity Analysis

- **Preprocessing:**
  - Sorting edges: $O(m \log m)$.
  - Kruskal tree construction: $O(m\alpha(n))$ for DSU operations plus $O(n \log^2 n)$ for small-to-large set merging (each color moved $O(\log n)$ times at $O(\log n)$ cost per move).
  - Binary lifting table: $O(N \log N) = O(n \log n)$.
  - Total: $O(m \log m + n \log^2 n)$.
- **Queries:**
  - Each query: $O(\log n)$ via binary lifting.
  - q queries: $O(q \log n)$.
- **Overall:** $O(m \log m + n \log^2 n + q \log n)$.