Trading off recourse budget with query times

- Interval additions: O(k2 log3 n) amortized time.
- Queries for a color of a particular interval: O(logn) time.
- Data structure: Modified interval tree.
- Input: k colorable interval graph
- For each node v, they will store:
  - lv → vertical line they intersect,
  - Sv → subset of intervals which intersects lv line,
  - Tv → subtree,
  - Iv → union of all Su's of nodes of Tv.

- Properties of T:
  - If $I_v = \emptyset$ then v is a leaf of T with undefined value l_v and empty set S_v.
  - Otherwise, T_v has defined value of l_v, and
    - Left subtree: $T_x \rightarrow l_u$ values all nodes u of T_x are smaller than l_v.
    - $S_v = \{ I \in I_v \mid beg(I) \leq l_v \leq end(I) \}$
    - $I_x = \{ I \in I_v \mid end(I) < l_v \}$
    - No interval from the left subtree of T_v overlaps with an interval from the right subtree of T_v.

- Initialization:
  - For root, $l_r = 0 \rightarrow S_r = \emptyset$.

- We will make sure if $S_v = \emptyset$, $v != r$
  - Then v is a leaf node
    - This implies number of nodes are at max I.

- Give permutation to each edge.

- Calculation of color of interval I.
  - Find its index i in the node storing it, let say v.
  - Let $e_1, e_2, \ldots, e_h$ be the sequence of edges of P_v, e_1 is connected to root.
  - $\sigma_{e_h} = (\tau_{e_h} \circ \cdots \circ \tau_{e_2} \circ \tau_{e_1})$
  - Color of $I_i \in S_v$ is $\sigma_{e_h}(i)$

- If tree is balanced then O(logn), so maintain n(u) for each node.

Updating the Interval Tree - Insertion

- Search for position of Inew in T
  - If Inew ∈ Sv, then increase variable n(u) for all nodes along the path Pv
  - If no node found, then let w be the leaf of T at the end of the search path, then set lw = beg(Inew) and again update n(u) value along the path Pw.

- If tree becomes unbalanced at node u
  - Create subtree with intervals Iu
  - lu for the root node of subtree = median of all I′

- Terms wrt node u:
  - beg(u) = min { beg(I) | I ∈ Iu }, i.e. left-most point of any interval stored in Tu
  - end(u) → similar logic
  - Lu = {I ∈ I | beg(I) ≤ beg(u) < end(I)} \ Iu, i.e., intervals not stored in Tu but containing beg(u)
  - Ru → similar logic
  - Lu or Ru cannot be in u or at root
  - R - These edges go down toward the end_u boundary from below
  - L - These edges go up toward the end_u boundary from below
  - Both L and R set crosses the end_u

- Final Algorithm
  - Find position for Inew, if imbalance then rebuild the closest subtree, in this case u = w, otherwise u = v.
  - Find begu, endu, Lu, Ru along with the colors of Lu's and Ru's colors.
  - Keep colors of Lu same, greedily color Iu and Ru using K-colors.
  - Since Ru was K-colorable initially, there is a permutation $\mu \in per(k)$ mapping the old colors of Ru to its new colors.
  - Now permutation for edges e, stored in Pu or Tu may be changed as per follow
    - If e is not part of Lu or Ru → unchanged
    - $e \in L \rightarrow$ unchanged
    - $e \in R \rightarrow \sigma'(e) = \sigma(e) \circ \mu$
    - If e is part of Tu, then choose permutation such that σe induce the new colors of Iu.

Updating the Interval Tree - Deletion

- Search for position of Inew in T
  - If Idel ∈ Sv, then decrease variable n(u) for all nodes along the path Pv

- If tree becomes unbalanced at node u
  - Follow the same procedure which was followed for insertions.