We use a hash table $$M$$ to store matches, a hash table $$IC$$ to store ice creams and a augmented AVL tree to store ice creams.

Hash table $$M$$ maps key value match ID to specific match data. Every match stores two ice creams and their corresponding points from this match.

Hash table $$IC$$ matches ice cream ID to its corresponding average points and total matches competed.

The AVL tree is sorted by ice-cream's average points. The AVL tree is augmented in the way that every node stores the ID, total match competed, average points, and also the number of nodes in its left tree $$num\\_l$$ and right tree $$num\\_r$$.

$$record$$ and $$clear$$ are trivial operation on $$M$$, but they also need to update the AVL tree. They firstly find the original ice cream data by accessing $$IC$$ with ID, then update $$IC$$ table. Then remove the original node in AVL tree and insert a new node with updated information.

During remove, at a specific node, if the decision is going left subtree, $$num\\_l$$ decrease by 1 and if going right $$num\\_r$$ decrease by 1. Insert is similar, just increase by 1 before going to the corresponding subtree. $$maintain$$ is hard to describe with many scenarios but with the same logic we can always update $$num\\_l$$ and $$num\\_r$$ correctly. Additional operation (update number of nodes in subtree) takes constant time at each node, so the worst case time complexity is still the height of the tree which is O(logn).

With the additional information, we can find $$ranked\\_winner$$ with only one traversal in the AVL tree from the root, which takes O(logn).

ranked\_winner(k):

Step 1: at root node, $$cur\\_node = root$$, $$cur\\_rank= cur\\_node .num\\_r+1$$

Step 2:

If k < cur\_rank, $$cur\\_node = cur\\_node.right \quad cur\\_rank = cur\\_rank - cur\\_node.num\\_l$$

if k > cur\_rank, $$cur\\_node = cur\\_node.left \quad cur\\_rank = cur\\_rank + cur\\_node.num\\_r$$

if k = cur\_rank, return $$cur\\_node.ID$$

repeat step 2