Cartesian trees, also known as Treaps — Tutorial

Gleb Evstropov, Andrew Stankevich

September 27, 2019

Problem Tutorial: "The Sum Again"

Implement explicit key Cartesian tree. In each node you should store the sum of the elements in the subtree. To process the query, cut away the query segment and print the stored sum value of the root node.

Problem Tutorial: "K-th Maximum"

Simply implement the Cartesian tree and store the size of the subtree in every node. To answer the query traverse down from the root, going left or right, whether there are enough nodes in the left subtree.

Also, as the problem is offline (i.e. all queries are known at the beginning) we can replace Cartesian tree with the segment tree.

Problem Tutorial: "Range Minimum Query"

Implement implicit key Cartesian tree. In each node you should store the size of the subtree and the minimum element in the subtree. To process the query, cut away the query segment and print the stored min of the root node.

Problem Tutorial: "Move to Front"

Implement implicit key Cartesian tree. In each node you should store the size of the subtree. To process the query, cut away the segment and merge it to the front of the tree. To print the answer, make left-root-right traversal of the resulting tree.

Problem Tutorial: "Reverses"

Implement implicit key Cartesian tree. In each node you should store the size of the subtree, the sum of elements in the subtree, and the flag whether the segment that is represented by this subtree should be reversed.

When propogating information about reverses down in "split" and "merge" operations, if the reverse flag in the vertex is set, you should exchange left and right subtrees of the current vertex, and invert their reverse flags.

To process the reverse query, cut away the query segment and change the flag in the root.

To process the sum query, cut away the query segment and print the stored sum value of the root.

Problem Tutorial: "Swapper"

Keep two separate Cartesian trees, one for even indices and one for odd.

To swap even and odd positions, cut away corresponding segments from the trees for odd and even positions, insert the segment from the tree of odd to the tree of even and vice versa.

To access the element just query the corresponding position in the corresponding tree.

Problem Tutorial: "Database"

Consider the elements in sorted order. Operation of the first type is equivalent to flipping the diagram with respect to the line $x = y$. Two consecutive operations of type one remain the sequence unchanged.

You should keep two versions of the Cartesian tree, one for sorted order by rows, and another for sorted order by columns, and apply operations carefully.

Problem Tutorial: "Permutations Strike Back"

Implement some kind of two-dimensional segment tree. Let us build a segment tree with a Cartesian tree in each node. The Cartesian tree in the each node of the segment tree stores all elements of the subtree, thus the total size of the segment tree is $O(n \log n)$ (each element is stored on each level exactly once).

To change the element, consider all Cartesian trees that contain this element. There are $O(\log n)$ such trees, one operation takes $O(\log n)$, thus the total worktime of one change operation is $O(\log^2 n)$.

To query the number of element between l and r with values between x and y, split the segment $[l, r]$ to $O(\log n)$ segments that are nodes of the segments tree. Consider the Cartesian tree in each segments and ask the number of element between x and y. This gives complexity $O(\log^2 n)$ per one query.

Problem Tutorial: "Key Insertion"

Store continuous filled segments in Cartesian trees with implicit key. Also keep DSU of segments, that would allow to query which segment each memory cell belongs. DSU should also store the leftmost index of the segment.

To insert the value, just insert it to the corresponding Cartesian tree, and merge with the cell to the right of the segment. Should the segment touch another segment, union them in DSU, and merge corresponding Cartesian trees. To find the position to insert to, subtract the index of the cell affected and the leftmost index of the segment.

Additional care is needed to process cells that are yet unoccupied, because it can lead to three segments merging.

Problem Tutorial: "Amber Ball"

We need to build the graph that for each segment i denotes whether the ball will fall to the ground or the index of first segment on its path after segment i.

Run a scanline from left to right, keep the current order of obstacles for the given vertical line. Relative order of two obstacles never changes because they do not intersect. While adding an obstacle or removing it from the Cartesian tree we can use the current x position in comparator.

When we remove an obstacle query the first down to it, thus obtain the outgoing edge in the graph. When the position of the scanline matches one of the queries, the first obstacle in the Cartesian tree will be the first on the path of the ball.