Problem Tutorial: "Intellectual Implementation"

Let's make a graph where we will connect two indices if corresponding rectangles intersect. Then, we need to compute number of anti-triangles (i. e. triples of vertices that are pairwise not connected with an edge) in this graph. Instead of doing this we will do three things:

1. Compute the degree of each vertex (for each rectangle we need to know how many other rectangles does it interesect).
2. Compute the number of triangles (the number of triples of rectangles which intersect pairwise).
3. Compute the answer to the original problem from this two values.

We will do 1 using sweepline. We will maintain a structure which helps us to answer the following queries:

1. Add a segment to the set.
2. Delete a segment from the set.
3. For given segment, calculate how many segments from the set it intersects.

This structure can be easily implemented using segment tree or Fenwick tree, since we can notice that answer to the third query for segment $[l, r]$ is equal to number of segments in a set minus number of segments with right border ¡ l minus number of segments with left border ¿ r. So we maintain two segment trees (or Fenwick trees) for left and right borders, then queries are equivalent to point update and range query.

After this, let's sort our rectangles by l. Then, we will solve two independent problems:

1.1) Count the number of such j that $l_i > l_j$ and rectangle i intersects with rectangle j.

1.2) Count the number of such j that $l_i < l_j$ and rectangle i intersects with rectangle j.

Sum of answers for 1.1 and 1.2 will be degree of vertex i.

Note that for 1.1, for rectangle j, to intersect with i, we must have $l_i < l_j < r_i$ and $[d_i, u_i]$ intersecting with $[d_j, u_j]$. So we can iterate in the increasing order of x coordinate, add $[d_i, u_i]$ to set at the moment l_i. Then answer for i will be the number of segments intersecting with $[d_i, u_i]$ at the moment r_i minus number of segments intersecting with $[d_i, u_i]$ at the moment l_i.

1.2 is done in a similar way, but we should add segment to the set at the moment l_i and delete it at the moment r_i (since in this case we have the condition $l_j < l_i < r_j$).

To do 3 let's compute the following number — number of triples (x, y, z) such that (x, y) and (x, z) are both connected or both not connected with an edge. Here we suppose that order of y and z doesn't matter. We can count it in two ways. If we fix x and suppose that $deg_x = d$ than we should add $\binom{d}{2} + \binom{n-1-d}{2}$. On the other hand, we can notice that triangle and anti-triangle triples contribute 3 to this value and all other triples contribute 1. Since you know total number of triples(namely, $\binom{n}{3}$), you can find the final answer.

Now we are left with 2. For this, let's also do sweepline over x coordinate. Suppose that we are able to answer the following queries:

1. Add a segment to the set.
2. Delete segment from the set.
3. Find how many triples of segments intersect.

If we will be able to do this, subproblem 2 will be also solved easily — we will just iterate over x coordinate and for rectangle with X-segment $[l, r]$ we will add its Y-segment $[d, u]$ to the set at moment l and delete it from the set at moment r. Also, we will calculate the difference of number of intersecting segments before adding it and after adding it. Sum of this differences over all the rectangles will be equal to total number of intersecting triples of rectangles(basically each intersecting triple will be taken into account for the rectangle with the biggest value of l). So we are left with the problem of processing these queries.

To do this, let's suppose that we will maintain two arrays — val and val'. $val[i]$ will be equal to number of segments $[l, r]$ in a set, such that $l \le i \le r$. $val'[i]$ is similar, but we will have $l \le i < r$. Then, answer to 3 is equal to $\sum \binom{val[i]}{3} - \binom{val'[i]}{3}$.

Then, our following is equivalent, to the following one

1. Do range add query (in our problem, it's either $+1$ or -1 but it doesn't matter).
2. Calculate $\sum \binom{val[i]}{3}$.

To solve this final problem, we will maintain a segment tree. In each node we will store $\sum \binom{val[i]}{z}$, for $0 \le z \le 3$. The only thing that we need to do, is to be able to do lazy updates, so we should be able to compute $\sum \binom{val[i]+lazy}{t}$. We can do this using Vandermonde's Identity:

$$\binom{val[i] + lazy}{t} = \sum_{k=0}^{t} \binom{val[i]}{k} \cdot \binom{t}{k}.$$

It's also nice that it works even for negative values of lazy (if we expand the definition of binomial coefficients).

So, we can do lazy propagation operation in constant time. Merge operation is also trivial.

Finally, problem is solved in $O(nlog(n))$ time (with pretty high constant though).

One last note is that actually you can solve queries with maintaining $\sum \binom{val[i]}{z}$, for $0 \le z \le 2$ (since you just need to calculate how many pairs of segments intersect with $[l, r]$). If you maintain $\sum \binom{val[i]}{3}$, be aware of integer overflow.