Hints for the Square Problem

This note contains ideas for solving the "small size" inputs for the `square` programming problem.

The SegTree data structure from lecture can efficiently maintain a collection of variables $a[0], \ldots a[n-1]$ (initially all zero) under the following operations.

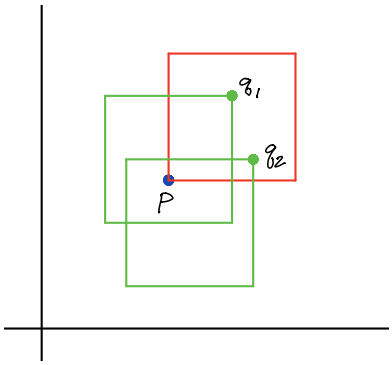
Assign(i, x):	Execute the assignment $a[i] \leftarrow x$.
RangeSum(i, j):	Return $\sum_{i \le k \le j} a[k]$.

Suppose instead that we could support these operations:

AddToRange(i, j, Δ):	$\forall k \in \{i, i+1, \ldots, j\}$ do $a[k] \leftarrow a[k] + \Delta$.
GlobalArgMax():	Return $(\max_{0 \le k \le n-1} a[k], \text{argmax}_{0 \le k \le n-1} a[k])$,

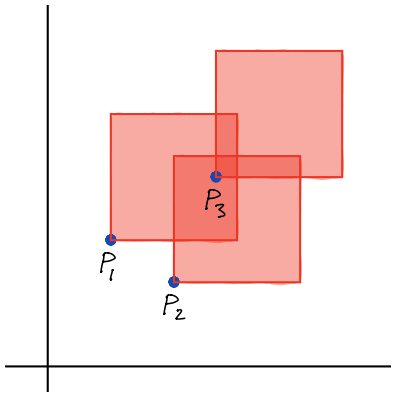
Later on we'll discuss how to create SegTrees that support these operations in $O(\log n)$ time and $O(n)$ space. For the moment let's talk about how this can help you solve the `square` problem.

There's a key observation about this problem that is going to be needed to get a fast weep-line algorithm. It's illustrated in the following diagram.



The blue dot p is one of the input points to the problem. The red square is the square of side s whose lower left corner is at p. This square has the following property: This square contains precisely all points q such that if we draw a green square whose upper right corner is at q that green square contains the blue dot p. This is an if and only if statement. Two such green squares are illustrated in the diagram.

Now suppose there are three input points p_1, p_2, and p_3 located as shown in the following diagram.

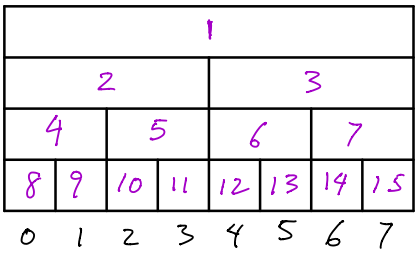


Imagine that when we processed point p_1 we raised the "height" of the square into the third dimension by 1. And we added an additional 1 to the height for all those in the other two squares. So every point in the plane has a height of 0, 1, 2, or 3. A point q with height i has the property that a square whose upper right corner is at q contains i of the points.

This leads immediately to a sweep-line algorithm based on the "AddToRange" SegTree described above.

As we sweep from left to right we first come to $p_1 = (x_1, y_1)$. We then increment the range $[y_1, y_1 + s]$. We do the same for p_2 and p_3. When we come to $x_1 + s$ we decrease the range $[y_1, y_1 + s]$ by 1 as the square defined by p_1 leaves its region of influence. Notice that at any time in this process GlobalArgMax() tells us the maximum height, and the y value where it happens. Specifically after p_3 is inserted we see that the height is 3. This is the highest ever achieved, so that tells us that the maximum number of points that can be contained in a square is 3. From this information we can also derive where that square is.

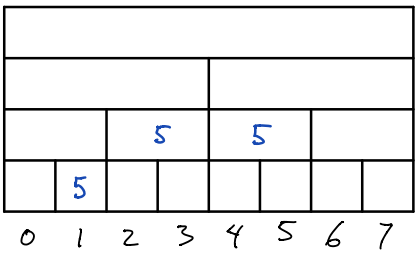
Now we return to the question of how to implement the "AddToRange" SegTree that we need. Here's a segtree, where the black numbers along the bottom represent the "user-facing" indices for the variables $a[0], \ldots, a[7]$ that we're representing.



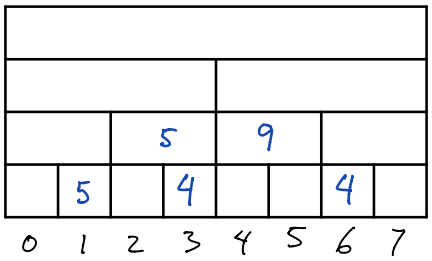
The purple numbers are the indices into the array of length 16 used to store the segtree. Let's call these the boxes.

Now we're going to modify (compared to the lecture) what is in the boxes. (In lecture a box stored the sum of all the variables below it, so the value of a variable was stored in the boxes on the lowest level.) Here the value of a variable will be represented by the *the sum* of all the "delta" values in the boxes above it. So, the value of $a[3]$ is the sum of the contents of boxes 11, 5, 2, and 1. This will allow us to very efficiently implement the AddToRange operation.

Consider the following state (empty boxes contain 0).



This means that all of $a[1], \ldots a[5]$ have a value of 5. Now suppose I wanted to do AddToRange$(3, 6, 4)$. All I have to do is add 4 to the contents of boxes 11, 6 and 14, as shown below.

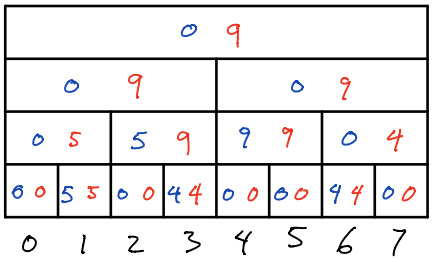


The set of boxes that must have their values adjusted in this operation is the same set that are summed in the RangeSum operation. So the code from lecture that does that can be repurposed for this.

The only thing left is to handle GlobalArgMax(). This is easily done by adding one more

field to each box. Namely the box stores the maximum value occurring in the given range *not considering the adjustments coming from the boxes above.*

So, completing the above example, the following diagram shows that "delta" values of the boxes in blue, and the "max" values in red.



The max value of a box b is the delta of box b plus the maximum of the maxs of the two child boxes. This allows it to be updated efficiently whe processing an AddToRange operation. It's also not hard to figure out how to do the GlobalArgMax().

You should be able to figure out the remaining details yourself. For the larger inputs some new ideas are needed.