CS634 Homework 1

Qilin Ye

September 22, 2025

Solution to problem 1. We assume all rectangles are bounded for otherwise the answer is clearly infinity. Let $\ell(x)$ be the total length of the intersection of the vertical line at x with U, the union of all given rectangles.

Because all rectangles are orthogonal, $\ell(x)$ can only change when x crosses a rectangle's vertical side. To this end we sort the distinct x-coordinates of all vertical sides of all rectangles, and observe that the area of U is then $\sum_k \text{value}(\ell(x))(x_{k+1} - x_k)$.

To faithfully record the value of $\ell(x)$ we will instead build a tree on the y-values. Sort the distinct y-endpoints $y_1 < \cdots < y_m$ and form the intervals $I_k = [y_k, y_{k+1})$. Build a segment T whose leaves correpsond to these I_k in the sorter order. Every node v represents a contiguous y-interval $J(v)$ by merging its descendants; in addition each node v stores $|J(v)|$, the number $c(v) \geqslant 0$ of rectangles that cover all of v, as well as a covered length $L(v) \geqslant 0$ of how much of v is covered by at least one rectangle.

Observe that

$$L(v) = \begin{cases} |J(v)| & c(v) > 0 \\ 0 & c(v) = 0 \text{ and } v \text{ is a leaf} \\ L(\text{left}(v)) + L(\text{right}(v)) & c(v) = 0 \text{ and } v \text{ is internal.} \end{cases}$$

For each rectangle $R_i = [x_i^-, x_i^+] \times [y_i^-, y_i^+]$ create two events: insert $[y_i^-, y_i^+)$ at x_i^-, and delete the same interval at x_i^+. Sort all events by x and process them at increasing, distinct x-values (if duplicates, process in batches). Set $A = 0$ and $x_{\text{prev}} = x_1$.

Iteratively, for each x_k, accumulate $A \leftarrow A + L(\text{root}) \cdot (x_k - x_{\text{prev}})$ because $L(\text{root})$ is the current vertical union length, and the quantity increases by the area of the slab $[x_{\text{prev}}, x_k)$. Then, apply all updates whose x-corodinate equals x_k by adding ± 1 to $c(v)$ on the nodes that partition $[y_i^-, y_i^+)$. When this is done, set $x_{\text{prev}} \leftarrow x_k$ and move on to next iteration. Finally, return A. Doing so allows us to calculate the total area by accumulating the area of each slab, as observed earlier.

For runtime, sorting takes $\mathcal{O}(n \log n)$ for both x- and y-endpoints. Each of the $\leqslant 2n$ updates touches $\mathcal{O}(\log n)$ nodes with $\mathcal{O}(1)$ work per node, so the total time is $\mathcal{O}(n \log n)$ and space $\mathcal{O}(n)$.

Solution to problem 2. Let $e_1 \leqslant e_2 \leqslant \cdots \leqslant e_{2n}$ be the sorted list of all endpoints of the n intervals (break ties arbitrarily). We construct intervals δ_i, $1 \leqslant i \leqslant m$, greedily from left to right, making each half-open $[a_i, b_i)$ with $b_i = a_{i+1}$.

Fix left boundary $a = e_1$. For $b > a$, consider $f(b) := |L([a,b])| - |S([a,b])|$. Clearly, the value of b can only change at endpoints e_i. Three possibilities as we increase b and hit some e_i:

- e_i is a left endpoint. Then S increases and L stays unchanged. Hence f decreases by 1.
- e_i is a right endpoint of an interval I that contains a. Then S increases by 1 and L decreases by 1, since I was previously long, but now short in $[a, b)$. Hence f decreases by 2.
- e_i is a right endpoint of an interval I that doesn't contain a. Nothing changes.

Thus $f(\cdot)$ is an integer-valued, monotonically decreasing function, with $f(a) = |\{I : a \in I\}| \geqslant 0$. Let $b > a$ be the smallest endpoint to the right of a for which $f(b) \leqslant 1$, and set $\delta = [a, b)$. Because each jump is at most -2, we see that $f(b) \in \{0, 1\}$, which is what the problem requires, so we add $[a, b)$ to Δ. Now reset $a \leftarrow b$ and repeat until the rightmost endpoint e_{2n} is hit. Finally, add $(-\infty, e_1)$ and $[e_{2n}, \infty)$ to the collection. This collection has $m = \mathcal{O}(n)$ intervals.

Now we build a data structure based on Δ. For each $\delta_i = [a_i, b_i)$, define disjoint sets of intervals:

- $L_i = \{I : \delta_i \subset I\}$ (long), stored as any list;
- $A_i = \{I : a_i \leqslant \ell(I) < b_i \leqslant r(I)\}$ (left endpoint in δ_i only), sorted by left endpoints $\ell(I)$;
- $B_i = \{I : \ell(I) \leqslant a_i < r(I) < b_i\}$ (right endpoint only), sorted by right endpoints $r(I)$;
- $C_i = \{I : I \subset \delta_i\}$, stored via a local stabbing structure like an interval tree so that the count query takes $O(\log|C_i|)$ time.

For each interval $I \in \mathcal{I}$, looking at the cell that contains its left endpoint $\ell(I)$, it can contribute exactly once between some A and some C. Looking at its right endpoint $r(I)$, it can contribute exactly once between some B and some C. Finally, it contributes to every $\delta_i \subset I$. Since $||L_i| - |S_i|| \leqslant 1$ for each i, and because each endpoint lies in exactly one cell,

$$\sum_i |A_i| + \sum_i |B_i| + \sum_i |C_i| = \sum_i |S_i| \leqslant 2n \qquad \text{and} \qquad \sum_i |L_i| \leqslant \sum_i |S_i| + m = \mathcal{O}(n),$$

a linear-size data structure as required.

Preprocessing first involves sorting endpoints, $\mathcal{O}(n \log n)$. Building Δ and populating A_i, B_i, C_i can be done in linear time by sweeping. Building the local trees on C_i take $\sum_i \mathcal{O}(|C_i| \log |C_i|) \leqslant \mathcal{O}(n \log n)$, so overall everything takes $\mathcal{O}(n \log n)$ combined in the preprocessing step.

Finally, in a query for x, we should locate the cell δ_i containing x. Clearly we report all intervals in L_i. For A_i, binary search the last $\ell \leqslant x$ and output the prefix (so every reported interval has $r(I) \geqslant b_i > x$). For B_i, binary search for the first $r \geqslant x$ and report the suffix (so every reported interval has $\ell(I) \leqslant a_i \leqslant x$). Finally, for C_i, query the local structure on C_i w.r.t x to report all $I \in C_i$ with $x \in I$. This runtime is $\mathcal{O}(\log n + k)$ as stated.

Solution to problem 3. (i) Pankaj said kd trees and $O(n^{2/3} + k)$ screams $k = 3$. So we up-project the said special triangles to $\mathbb{R}^3$ and turn them into axis-aligned boxes via $(x, y) \mapsto (x, y, x + y)$. Then a point lies in the triangle iff it lies in the axis-aligned 3D box.

Formally, a triangle of the said shape is defined by lines $x = a_x, y = a_y$, and $x + y = a_s$ for some a_x, a_y, and a_s. The region enclosed can be defined by

$$\Delta = \{(x, y) : x(\sigma_x)a_x, y(\sigma_y)a_y, (x + y)(\sigma_s)a_s\}$$

where $\sigma_x, \sigma_y, \sigma_x \in \{\leqslant, \geqslant\}$ define the orientation of the triangle. The up-projection is defined by

$$\Delta \mapsto B = I_x \times I_y \times I_s = \{x : x(\sigma_x)a_x\} \times \{y : y(\sigma_y)a_y\} \times \{s : s(\sigma_s)a_s\}$$

where $s = x + y$. Then clearly $(x, y) \in \Delta$ iff $(x, y, x + y) \in B$.

For (i), we build a 3D kd-tree on the set $\{(x_i, y_i, x_i + y_i)\}$. Each node stores the bounding box of its subtree. As described in lecture, this process takes space $\mathcal{O}(n)$ and time $\mathcal{O}(n \log n)$.

For query, given a triangle Δ, build the three intervals I_x, I_y, I_s as described and form the box B. Run the standard kd-tree range search on B, as described in lecture: prune nodes disjoint from B, report entire subtrees whose cell is contained in B, and recuse otherwise. Runtime is $\mathcal{O}(n^{2/3} + k)$ as discussed in lecture.

(ii) We build a 3D range tree on $\{(x_i, y_i, x_i + y_i)\}$ instead. We build a balanced BST T_x using the x-coordinates. Then, at each node T_x, store a 2D range tree $T_v^{(y,s)}$ on pairs (y, s) from v's subtree. Apply fractional cascading in each $T_v^{(y,s)}$ so that after locating a y-range in one node, the search in the associated lists at neighboring nodes cost only $\mathcal{O}(1)$ extra each. This gives the desired space and runtime complexities.

Solution to problem 4. The tree we build is basically analogous to the ones in the problem Sort the $2n$ x-coordinates of all endpoints and partition $\mathbb{R}$ into left-closed, right-open intervals like in the previous problems. For each slab σ, let $A(\sigma)$ be the number of segments covering the slab. Sweep a vertical line from left to right over the endpoints and maintain a BST T of an active set, keyed by the segment y-coordinates and threaded by in-order successor. When we encounter a left endpoint, we insert the segment; when we encounter a right endpoint we delete it. To make T persistent, we store the root point after each event, and make this root the version T_σ for the next slab. Every update changes $O(1)$ pointers, so across $\mathcal{O}(n)$ events the total space needed is $\mathcal{O}(n)$.

Now for query. Given a point $q = (q_x, q_y)$, binary search the endpoint list to locate the slab σ containing q_x. Fetch the pointer T_σ. Inside T_σ, perform a search to find the smallest key $\geqslant q_y$, which takes $\mathcal{O}(\log n)$ time. From that node, report it and then follow successor links, outputting all subsequent nodes. Each of them corresponds to a segment $s \in A(\sigma)$ with $y(s) \geqslant q_y$, hence intersecting the upward ray from q and is a segment of interest. This gives the desired runtime.

Solution to problem 5. Sort the points by x-coordinates. For each color c, let the x-coordinates of the k_c points with color c be sorted into $x_1^c < \cdots < x_{k_c}^c$, and form exactly h_c horizontal segments

$$s_i^c = (x_{i-1}^c, x_i^c] \times \{-x_i^c\}, \qquad x_0^c = -\infty, \qquad i \in 1, \cdots, k_c.$$

By doing so, we have created a series of segments corresponding to a fixed color that also partition the line, such that for any x-value, there is at most one active segment of that color. Given a vertical sweeping line at x, the right endpoint of the intersecting segment x_i^c is precisely the smallest x_i with color c to the right of x.

Now repeat the process for each color and let the resulting union be S.

Now we consider a query for $I = [a, b]$. Observe: a color c appears in I iff the first occurrence of c to the right of a is at a position $\leqslant b$ (otherwise, the smallest x_i^c is $> b$ and therefore does not intersect with I). By construction, the segment of a color c active at $x = a$ lies at height $y = -x_i^c$, and the condition $x_i^c \leqslant b$ is equivalent to $-x_i^c \geqslant -b$, so now we have reduced the problem into Q4: the colors in I are in one-to-one correspondence with the segments of S that intersect the vertical ray starting at $(a, -b)$ and going upward. Apply the data structure of Q_4 to S and

query with point $(a, -b)$. Report, for each returned segment, its colored label. Runtime analyses follow directly from Q4. Preprocessing (sorting) takes $\mathcal{O}(n \log n)$, whereas building S takes $\mathcal{O}(n)$ space and time. A query, as in Q4, takes $\mathcal{O}(\log n + t)$, as required.