Assignment 8

Sample solutions

CSCI 3110 — Summer 2018

Throughout, this solution, I will use h to refer to horizontal segments and v to refer to vertical segments. The y coordinate of a horizontal segment h is $h.y$; the x-coordinates of its endpoints are $h.x_1$ and $h.x_2$. Similarly, $v.x$, $v.y_1$, and $v.y_2$ refer to the x coordinate and the two y-coordinates of the endpoints of the vertical segment v.

(a) Two segments h and v intersect if and only if $h.x_1 \leq v.x \leq h.x_2$ and $v.y_1 \leq h.y \leq v.y_2$. Since the question states that $h.x_1 \leq v.x \leq h.x_2$ for every horizontal segments h and every vertical segment v, we only need to identify all pairs (h, v) such that $v.y_1 \leq h.y \leq v.y_2$. Let us refer to the segments in V as $v_1, \ldots, v_{n_1}$ and to the segments in H as $h_1, \ldots, h_{n_2}$, where $n_1 + n_2 = n$.

The algorithm. The algorithm maintains two indices i and j such that v_i is the "current segment" in V and h_j is the "current segment" in H. Initially, $i = 1$ and $j = 1$. As long as $i \leq n_1$, we do the following:

1. While $j \leq n_2$ and $h_j.y < v_i.y_1$, increase j by one. After this, we either have $j > n_2$ or $h_j.y \geq v_i.y_1$.
2. Set $k = j$. While $k \leq n_2$ and $h_k.y \leq v_i.y_2$, increase k by one. After this, we either have $k > n_2$ or $h_k.y > v_i.y_2$.
3. If $j \leq n_2$, output the pairs $(v_i, h_j), \ldots, (v_i, h_{k-1})$ as intersecting pairs of segments.
4. Increase i by one.

Correctness. To prove the correctness of the algorithm, we need to argue that the algorithm outputs a pair (v_a, h_b) if and only if v_a and h_b intersect. First assume that (v_a, h_b) intersect. Then $v_a.y_1 \leq h_b.y \leq v_a.y_2$. If at the beginning of the ath iteration of the loop through V, we have $j \leq b$, then step 1 does not increase j above b because $h_b.y \geq v_a.y_1$. As pointed out, we have $h_j.y \geq v_a.y_1$ at the end of step 1. Step 2 now finds the smallest value $k \geq j$ such that either $k > n_2$ or $h_k.y > v_a.y_2$. Since $h_b.y \leq v_a.y_2$, we have $k > b$. Now Step 3 outputs all pairs $(v_a, h_j), \ldots, (v_a, h_{k-1})$ as intersecting. Since $j \leq b < k$, this includes the pair (v_a, h_b).

Is it possible that $j > b$ at the beginning of the ath iteration? If so, then there exists an index $i < a$ such $j \leq b$ at the beginning of this iteration and $j > b$ at the end of this iteration. This implies that $h_b.y < v_i.y_1$, a contradition because the segments in V are sorted by their bottom endpoints, that is, $v_i.y_1 < v_a.y_1$ but $v_a.y_1 \leq h_b.y$.

Now assume that (v_a, h_b) do not intersect. Then either $h_b.y < v_a.y_1$ or $h_b.y > v_a.y_2$. If $h_b.y < v_a.y_1$, Step 1 of the ath iteration of the outer loop ensures that $j > b$. If $h_b.y > v_a.y_2$, then $h_b.y > v_a.y_1$. As argued in the case when (v_a, h_b) intersect, this implies that $j \leq b$ at the beginning of the ath iteration of the loop through V. Since Step 2 increases k only until $h_k.y > v_a.y_2$, this

implies that $k \leq b$. Thus, we either have $b < j$ or $b \geq k$; in either case, (v_a, h_b) is not part of the list of intersections reported for v_a.

Analysis. For each iteration of the loop through V, step 4 takes constant time while step 1 takes constant time plus constant time for every increase of j. Since j never decreases and cannot exceed n_2, this shows that total cost of steps 1 and 4 is $O(n_1 + n_2) = O(n)$. In the ith iteration of the loop through V, the cost of steps 2 and 3 is $O(1 + k_i)$, where k_i is the number of reported intersections. Summing this over all iterations, the cost of steps 2 and 3 is $O(n + k)$. Thus, the algorithm takes $O(n + k)$ time in total.

(b) Given the slab $S = [x_\ell, x_r] \times (-\infty, +\infty)$ and y-sorted lists V and H of all vertical and horizontal segments with at least one endpoint in S, we find all intersections between segments in V and H as follows:

First, we collect the x-coordinates of all endpoints of segments in H and V, ignoring endpoints that do not lie in S. This clearly takes linear time. Using the linear-time selection algorithm, we can find the median x_m of these coordinates and define two new slabs $S_\ell = [x_\ell, x_m] \times (-\infty, +\infty)$ and $S_r = [x_m, x_r] \times (-\infty, +\infty)$. By scanning H and V, we can construct lists H_ℓ, V_ℓ, H_r, and V_r, where V_ℓ and V_r contain the vertical segments in V contained in S_ℓ and S_r, respectively, and H_ℓ and H_r contain the horizontal segments in H with at least one endpoint in S_ℓ and S_r, respectively. Note that a segment in H may end up in both H_ℓ and H_r if the segment has one endpoint in S_ℓ and one endpoint in S_r.

If two segments $h \in H$ and $v \in V$ intersect, assume w.l.o.g. that $v \in V_\ell$. If $h \in H_\ell$, we can find the intersection by calling the algorithm recursively on the input (S_ℓ, V_ℓ, H_ℓ). We call this a type-I intersection. If $h \notin H_\ell$, then h has no endpoint in S_ℓ. Since it intersects v, it must intersect S_ℓ, which is possible only if $h.x_1 < x_\ell$ and $h.x_2 > x_m$. We call this a type-II intersection.

This analysis immediately leads to the following divide-and-conquer algorithm: In addition to the lists H_ℓ, H_r, V_ℓ, V_r described above, we also produce lists H'_ℓ and H'_r containing all segments $h \in H$ such that $h.x_1 < x_\ell$ and $h.x_2 > x_m$ or $h.x_1 < x_m$ and $h.x_2 > x_r$, respectively. Every type-I intersection is between segments in H_ℓ and V_ℓ or between segments in H_r and V_r. We call our algorithm recursively on (S_ℓ, H_ℓ, V_ℓ) and on (S_r, H_r, V_r) to find these intersections. Every type-II intersection is between segments in H'_ℓ and V_ℓ or between segments in H'_r and V_r. For these intersections, since each segment $h \in H'_\ell$ satisfies $h.x_1 < x_\ell$ and $h.x_2 > x_m$ and every segment $v \in V_\ell$ satisfies $x_\ell \leq v.x \leq x_m$, the pair of lists H'_ℓ and V_ℓ satisfies the conditions of part (a), so we can use the algorithm form part (a) to find these intersections in linear time. Similarly, the intersections between segments in H'_r and V_r can be found using the algorithm from part (a). (This requires that these lists are sorted by the y-coordinates of segment endpoints. I'll discuss in part (c) how this is easily done in linear time.)

How long does this algorithm take? For this analysis, it will be useful to use n to refer to the number of segment endpoints in S rather than the total number of segments in $H \cup V$. Since every segment in $H \cup V$ has one or two endpoints in S, this changes n by at most a factor of 2.

Apart from the recursive calls on (S_ℓ, H_ℓ, V_ℓ) and (S_r, H_r, V_r), the algorithm spends linear time to find the median x-coordinate x_m and to partition H and V into $H_\ell, H_r, H'_\ell, H'_r, V_\ell, V_r$. Assuming we can do this while keeping these lists y-sorted, finding the intersections between segments in (H'_ℓ, V_ℓ) and (H'_r, V_r) takes $O(n + k')$ time, where k' is the number of intersections this reports. Finally, observe that, if n is the number of segment endpoints in S, then S_ℓ and S_r both contain $n/2$

segment endpoints. Thus, if $T(n,k)$ is the time the algorithm takes on an input with n segment endpoints and k intersections, we obtain the recurrence

$$T(n,k) = T\left(\frac{n}{2}, k_\ell\right) + T\left(\frac{n}{2}, k_r\right) + O(n+k'),$$

where k_ℓ and k_r are the numbers of intersections between segments in (H_ℓ, V_ℓ) and (H_r, V_r), respectively. We have $k = k_\ell + k_r + k'$. It is now easy to show that $T(n,k) = O(n \lg n + k)$. For $2 \le n < 4$, we have only a constant input and output size, so the algorithm takes constant time, which is upper bounded by $c(n \lg n + k)$ for a sufficiently large constant c. For $n \ge 4$, we have

$$\begin{aligned} T(n,k) &\le T\left(\frac{n}{2}, k_\ell\right) + T\left(\frac{n}{2}, k_r\right) + a(n+k') \\ &\le c\left(\frac{n}{2}\log\frac{n}{2} + k_\ell\right) + c\left(\frac{n}{2}\log\frac{n}{2} + k_r\right) + a(n+k') \\ &= cn(\lg n - 1) + an + c(k_\ell + k_r) + ak' \\ &\le cn \lg n + ck \end{aligned}$$

as long as $c \ge a$.

(c) Now for filling in some missing minor details:

The initial slab that contains all segment endpoints can be chosen as $(-\infty, +\infty) \times (-\infty, +\infty)$. If you are uncomfortable working with infinity (which you would be if you were to implement this algorithm, you can instead scan the list of horizontal and vertical segments to find the x-coordinates x_ℓ and x_r of the leftmost and rightmost segment endpoints and choose $[x_\ell, x_r] \times (-\infty, +\infty)$ as the initial slab.

Before making the first recursive call, we need to ensure the segments in H are sorted by their y-coordinates and the segments in V are sorted by the y-coordinates of their bottom endpoints. This is easily done in $O(n \lg n)$ time using any optimal sorting algorithm. Thus, the cost of the algorithm remains $O(n \lg n + k)$.

Given the input lists H and V of an invocation on some slab S in y-sorted order, we produce $H_\ell, H_r, H'_\ell, H'_r, V_\ell, V_r$ by inspecting each segment in H or V and placing it into the appropriate subset of these lists. If we scan the segments in H and V in order and append each segment to $H_\ell, H_r, H'_\ell, H'_r, V_\ell, V_r$ as appropriate, this ensures that these lists are once again in y-sorted order, ready for the next recursive call or for the application of the algorithm from part (a).

The analysis of the running time of the algorithm was provided already as part of the answer to part (b).