

COMPUTATIONAL GEOMETRY

Homework Set # 8 – Solution Notes

(1). [20 points] *Problems 8.1, 8.8, BKOS.*

Problem 8.1: Let $p = (p_x, p_y)$ be a point and let $\ell : y = mx + b$ be a line, both in the “primal” plane. Then, $p^* : y = p_x x - p_y$ and $\ell^* = (m, -b)$.

Incidence: $p \in \ell \iff p_y = m \cdot p_x + b \iff -b = p_x \cdot m - p_y \iff \ell^* \in p^*$.

Order: p lies above $\ell \iff p_y \geq m \cdot p_x + b \iff -b \geq p_x \cdot m - p_y \iff \ell^*$ lies above p^* .

Problem 8.8: Let $p = (p_x, p_y)$ be a point and let $\ell : y = mx + b$ be a line, both in the “primal” plane. Consider a definition of duality in which $p^* : y = p_x x + p_y$ and $\ell^* = (m, b)$.

Incidence: $p \in \ell \iff p_y = m \cdot p_x + b$ and $\ell^* \in p^* \iff b = p_x \cdot m + p_y$. Now, it is easy to see that $p_y = m \cdot p_x + b$ is *not* equivalent to $b = p_x \cdot m + p_y$. Consider an example: $p = (1, 1)$ lies on line $\ell : y = 1x + 0$. The duals of these would be $p^* : y = x + 1$ and $\ell^* = (1, 0)$, and clearly ℓ^* is *not* on the line p^* .

Order: In the primal, p lies above $\ell \iff p_y \geq m \cdot p_x + b$. In the dual, ℓ^* lies above $p^* \iff b \geq p_x \cdot m + p_y$. Again, it is easy to see that $p_y \geq m \cdot p_x + b$ is *not* equivalent to $b \geq p_x \cdot m + p_y$. Consider an example: $p = (1, 2)$ lies above line $\ell : y = 1x + 0$. The duals of these would be $p^* : y = x + 2$ and $\ell^* = (1, 0)$, and clearly ℓ^* is *not* above the line p^* .

Thus, this duality does not preserve incidence or order.

(2). [20 points] *We have seen how to count the number of vertices, edges, and cells (2-faces) in an arrangement of lines in the plane. Now, let's consider a simple arrangement of planes in 3-space. (By “simple”, recall, that we mean that any three planes have a unique point in common, while any four planes have no point in common.) Derive formulas for the number of vertices, edges, facets (2-faces), and cells (3-faces) in a simple arrangement of n planes in 3-space.*

Let $f_k^{(d)}(n)$ denote the number of k -faces in a simple arrangement of n hyperplanes in d dimensions. In class, we showed that

$$f_0^{(2)}(n) = \binom{n}{2}, f_1^{(2)}(n) = 2\binom{n}{2} + n, f_2^{(2)}(n) = \binom{n}{2} + n + 1$$

Now, from the definition of a simple arrangement, we have

$$f_0^{(3)}(n) = \binom{n}{3}$$

If we sweep a vertical plane through the arrangement, we see that with each vertex we hit, three new edges begin. Initially, when the plane is at minus infinity, it crosses $\binom{n}{2}$ edges (one per pair of planes, since each pair intersects in a line). Thus,

$$f_1^{(3)}(n) = 3\binom{n}{3} + \binom{n}{2}$$

(Note that if you write a recursion, considering what happens when you add the n th plane, you get $f_1^{(3)}(n) = f_1^{(3)}(n-1) + f_0^{(2)}(n-1) + f_1^{(2)}(n-1)$, which solves to the same answer.)

Similarly, whenever the sweep plane hits a vertex, it creates three new 2-faces; initially, the sweep plane crosses $f_1^{(2)}(n)$ 2-faces at minus infinity, since the slice at minus infinity is just an arrangement of n lines. Thus, we get

$$f_2^{(3)}(n) = 3\binom{n}{3} + 2\binom{n}{2} + n$$

(Note that if you write a recursion, considering what happens when you add the n th plane, you get $f_2^{(3)}(n) = f_2^{(3)}(n-1) + f_1^{(2)}(n-1) + f_2^{(2)}(n-1)$, which solves to the same answer.)

Similarly, whenever the sweep plane hits a vertex, it creates one new 3-face; initially, the sweep plane crosses $f_2^{(2)}(n)$ 3-faces at minus infinity, since the slice at minus infinity is just an arrangement of n lines. Thus, we get

$$f_3^{(3)}(n) = \binom{n}{3} + \binom{n}{2} + n + 1$$

(Note that if you write a recursion, considering what happens when you add the n th plane, you get $f_3^{(3)}(n) = f_3^{(3)}(n-1) + f_2^{(2)}(n-1)$, which solves to the same answer.)

(3). [20 points] *Problem 8.16, BKOS. While the text asks for expected time/space bounds, you should be able to give deterministic time and space bounds.*

(a). Construct the arrangement (of size $O(n^2)$, in time $O(n^2)$) of the $2n$ lines dual to the $2n$ segment endpoints. Using DFS, walk through the faces of the DCEL and label each face with the number of double wedges (duals of segments) that cover that face (this number will go up or down as we cross from face to face, in a manner that is easily determined using local geometry and data). (This takes time linear in the size of the subdivision, which is $O(n^2)$.) Keep track, during the DFS, to see if there is a face that corresponds to being in *all* the double wedges; if such a face exists, this gives a stabber.

(b). If all the segments are vertical, then we are searching for a line, $y = mx + b$, that separates the “red” top endpoints from the “blue” bottom endpoints of each segment. Thus, it is a red-blue separation problem, which asks for the feasibility (in (m, b) -space) of finding a line for which the red points are above the line and the blue points are below the line. Thus, we can solve this 2-dimensional LP in worst-case time $O(n)$, since there are only $2n$ constraints.

(4). [20 points] *Problem 8.4, BKOS.*

It is easy to prove that the leftmost vertex of the arrangement, $\mathcal{A}(L)$, must be determined by two lines that are consecutive in angular order (slope order). (Draw a picture and consider cases!) Thus, a simple sort by slope gives an $O(n \log n)$ time algorithm for the leftmost vertex; the topmost, bottommost, rightmost are also defined by pairs of lines that are adjacent in slope order.

Also, find the flaw in the following solution, which a student once proposed:

Let $L = \{l_1, \dots, l_n\}$ be set of lines in the plane. Straightforward way of computing an axis parallel rectangle that contains all the vertices of the arrangement $\mathcal{A}(L)$ would be finding all the intersection points of all pairs of lines and finding minimal and maximal x and y coordinates. That takes $O(n^2)$, however.

How does the problem translate to dual plane? Imagine an axis-parallel bounding box that is big enough to contain all the vertices of the arrangement. Let's consider finding the lower bound in y -coordinate first. Lines in L in dual plane correspond to n points l_i^ . We can sort them by y -coordinate in $O(n \log n)$ and find the line C^* given by $y = M$ such that that all the points l_i^* lie below C^* . Since duality preserves order (point is below the line if and only if dual of the line is below the dual of the point), in primal plane point $C(0, -M)$ has to be below all of the lines l_i . Now take an intersection point p of two lines l_i and l_j . Since C is below both of the lines, it is in the specific “south” wedge out of 4 wedges formed by the two intersecting lines. Therefore y -coordinate of C has to be less than y -coordinate of the intersection point p . The same is true for any intersection point so we have found the lower y -coordinate bound. We can find upper y -coordinate bound for the rectangle at the same time (by taking the line D^* given by $y = m$ that lies above all the lines, which gives $-m$ as the upper bound).*

We can repeat the symmetrical reasoning and sort points in x -coordinate to find upper and lower limits (that will work in practice but theoretically we imagine rotating the coordinate system by 90 degrees to avoid dealing with vertical lines in dual plane that do not have original in primal plane).

Since the operations of computing the dual of line/point take constant time, the total running time of the algorithm is $O(n \log n)$.

Consider the claim “Since duality preserves order (point is below the line if and only if dual of the line is below the dual of the point), in primal plane point $C(0, -M)$ has to be below all of the lines l_i . Now take an intersection point p of two lines l_i and l_j . Since C is below both of the lines, it is in the specific “south” wedge out of 4 wedges formed by the two intersecting lines. Therefore y -coordinate of C has to be less than y -coordinate of the intersection point p .”

Just because C lies *BELOW* each of the two lines does not mean it lies below the intersection point: what if the two lines both have positive slope (or both negative slope)? Then you can easily draw a case in which C lies below both lines but lies much higher (in y) than the crossing point.

(5). [20 points] *Let L be a set of n lines in the plane such that no three lines pass through a common point. Assume that exactly k ($k \leq n$) of the lines are all parallel to each other (call this set L_0) and that all of the other $n - k$ lines (in the set $L_1 = L \setminus L_0$) are not parallel to each other or to any line of L_0 . Derive formulas for the number of vertices, edges, and faces (cells) in the arrangement, $\mathcal{A}(L)$, of the lines L . Your formulas for v , e , and f will depend*

on n and k . (Note that your formulas should match what we get for the case $k = 0$, when the set of lines forms a simple arrangement.)

Let $m = n - k$ be the number of lines in L_1 . Let k be an arbitrary nonnegative integer, and consider various values for m : $m = 0, 1, 2, \dots$. We let $v(m, k)$, $e(m, k)$, $f(m, k)$ denote the number of vertices, edges, and faces for any given choice of m and k .

Just to get a feel for the problem, we start with some small cases.

If $m = 0$, we have k parallel lines, so we have 0 vertices, k edges, and $k + 1$ faces. Thus, $v(0, k) = 0$, $e(0, k) = k$, and $f(0, k) = k + 1$.

If $m = 1$, we have k parallel lines and one other line that cuts across all k of the parallel lines, so we have k vertices, $2k + k + 1$ edges, and $2(k + 1)$ faces. Thus, $v(1, k) = k$, $e(1, k) = 3k + 1$, and $f(1, k) = 2k + 2$.

Method 1 Each of the m lines in L_1 intersects all k lines of L_0 and all $m - 1$ other lines of L_1 . Each of the k lines of L_0 intersects all m lines of L_1 .

Summing the number of crossings along each line gives $m \cdot (k + m - 1) + k \cdot m$. But this counts every crossing point *twice*. Thus, the number of crossing points is $v(m, k) = mk + m(m - 1)/2 = mk + \binom{m}{2}$. (In terms of k and n , this is $v = (n - k)k + \binom{n - k}{2}$.)

Each of the m lines of L_1 gets split into $(k + m - 1) + 1$ pieces by the $k + m - 1$ vertices along it; this gives $m \cdot (k + m)$ edges along lines of L_1 . Each of the k lines of L_0 gets split into $m + 1$ pieces by the m vertices along it; this gives $k \cdot (m + 1)$ edges along lines of L_0 . In total, we get $e(m, k) = m(k + m) + k(m + 1)$ edges. (In terms of k and n , this is $e = n(n - k) + k(n - k + 1)$.)

By Euler's formula, we get $f = e - v' + 2$, where $v' = v + 1$ is the total number of vertices *including the vertex at infinity* (where all of the infinite rays connect). Thus, $f(m, k) = e(m, k) - v(m, k) + 1 = m^2 + k(m + 1) - \binom{m}{2} + 1$.

Method 2 We can also solve this problem using recursion. We know the initial conditions: $v(0, k) = 0$, $e(0, k) = k$, and $f(0, k) = k + 1$. In general, consider the case of m lines in L_1 and consider what happens when we add the $(m + 1)$ st line, ℓ : We create $k + m$ new vertices, since ℓ crosses each of the existing lines. Along ℓ , we get then $k + m + 1$ new edges, and the $k + m$ edges that ℓ crosses each get split in two, adding $k + m$ more edges; the total increase in edges is therefore $2(k + m) + 1$. We will not worry about the number of faces, since we can get this from Euler's formula in the end anyhow.

For the vertices, we get the recurrence $v(m + 1, k) = v(m, k) + k + m$, with the initial condition that $v(0, k) = 0$, which solves to $v(m, k) = mk + m(m - 1)/2$.

For the edges, we get the recurrence $e(m + 1, k) = e(m, k) + 2(k + m) + 1$, with initial condition $e(0, k) = k$, which solves to $e(m, k) = m^2 + 2km + k$.

Euler's formula again gives the formula for the number of faces: $f(m, k) = e(m, k) - v(m, k) + 1 = m^2 + k(m + 1) - \binom{m}{2} + 1$.

(6). [20 points] *Problem 10.7(a), BKOS.* We want to solve the following query problem: Given a set S of n disjoint line segments in the plane, determine those segments that intersect a vertical ray running from a point $q = (q_x, q_y)$ vertically upwards to infinity. Describe a data structure for this problem that uses $O(n \log n)$ storage and has a query time of $O(\log n + k)$, where k is the number of reported answers. Explain how your method achieves the claimed bounds.

Store the segments S in a segment tree. Then, when we search the tree using q_x , we can simply walk through the list of segments stored at any given node of the segment tree, starting from the highest segment in the "strip" and ending at the last one that lies above q , reporting those segments we find in time proportional to their number.

(7). [20 points] *Problem 10.8, BKOS.*

We build a segment tree \mathcal{T} on the x -intervals of the rectangles in R . A node ν in \mathcal{T} corresponds to the vertical slab $\text{Int}(\nu) \times (-\infty : +\infty)$. A rectangle is in the canonical subset, $S(\nu)$, of ν if it completely crosses the slab corresponding to ν (it "spans" the slab) but not the slab corresponding to the parent of ν in \mathcal{T} .

We construct an associated data structure for each node ν of \mathcal{T} : We construct an interval tree on the y -intervals of the rectangles in $S(\nu)$.

When we search with a query point $q = (q_x, q_y)$ in \mathcal{T} , we access $O(\log n)$ canonical subsets corresponding to the nodes on the search path for q_x ; these canonical subsets collectively contain all of the rectangles whose x -interval contains q_x . A rectangle r in such a canonical subset, $S(\nu)$, contains query point q if and only if q_y lies in the y -interval of r ; this is determined by performing a search for q_y in the associated data structure (interval tree) for ν .

In total, a query consists of $O(\log n)$ searches (one for each canonical subset along the search path); one such search is done in time $O(\log n + k_\nu)$, where k_ν is the number of y -intervals reported for canonical subset $S(\nu)$. Since $\sum_\nu k_\nu = k$, the number of reported rectangles containing q , the total query time is $O(\log^2 n + k)$. The storage is $O(n \log n)$, since each level of the segment tree uses $O(n)$ storage (including the interval trees, whose storage is linear in the number of y -intervals they store). The preprocessing time is $O(n \log n)$ to build the segment tree on x -intervals, plus $\sum_\nu O(|S(\nu)| \log |S(\nu)|) = O(n \log n)$ for constructing the interval tree associated structures.

In higher dimensions, the data structure has multiple levels, much as we saw with range trees. Each dimension multiplies the space and the query search time by $\log n$, so we get $O(\log^d n + k)$ query time, $O(n \log^{d-1} n)$ space, and $O(n \log^{d-1} n)$ preprocessing time.