Solutions Homework 1 CPSC 516 2022

Dylan Brown

Seyed Ali Tabatabaee

1. Suppose we partition a simple polygon with n vertices into pieces by adding chords. Prove that the sum of the number of vertices in all of the pieces is $O(n)$.

   Proof. Consider an arbitrary partition P of an n-vertex polygon. If every region of P is a triangular region, then P is a triangulation and we will write $P = Q$. Otherwise, we can triangulate each of these non-triangular regions to make a partition Q which contains all the chords of P, plus some more.

   We have from lecture that every triangulation has $n-2$ triangles, each of which has 3 vertices, so Q has a total of $3n - 6$ vertices. Since P has at most as many vertices as Q it has $\leq 3n - 6 \in O(n)$ vertices, and we are done.

2. Without loss of generality, we assume no two points are on the same vertical line (We can always rotate all points so that this happens).

   We use a sweep line algorithm. Our sweep line is vertical and from left to right. It starts with line $x = x_{min}$ (x_{min} is the minimum x-coordinate among all points and triangle vertices) and ends with $x = x_{max}$ (x_{max} is the maximum x-coordinate among all points and triangle vertices). Our events are:

   - when a triangle starts
   - when a triangle ends
   - when we see a point

   Hence, the total number of events is $O(n)$.

   We keep a red-black tree structure, containing triangles that have started, but not ended yet and are not any other triangle (we call those triangles **alive**). The intersection of a triangle with a vertical line is either null (the triangle is not alive at that point), a point (start or end of the triangle), or an interval. For alive triangles, their intersections with the current position of the sweep line do not intersect (otherwise, it means that the triangles intersect which is impossible). Moreover, for the duration that any two triangles are alive together, the intersection of one of those triangles with the sweep line is always higher than the intersection of the other triangle with the sweep line and this order does not change (otherwise, it means that the two triangles intersect). Therefore, the order of alive triangles does not change and we have a consistent order for keeping them in the red-black tree structure. When our sweep line is at $x = cur$, for any alive triangle in the red-black tree we can compare in $O(1)$ if a point (cur, y) is inside, below, or higher than the interval of the alive triangle on the line $x = cur$.

   Now we see how to handle events:

   - When a triangle starts (it always starts with a point - let's call that point p), we check p with the alive triangles in the red-black tree. If p falls in the interval of any alive triangle, then the new triangle is inside an alive triangle and hence we don't need to care about the new triangle because if a point falls in it, it also falls in the alive triangle that we already have in the red-black tree. Hence, the new triangle will never become alive and won't be added to the red-black tree. If p is not inside the interval of any alive triangle, the new triangle will also become alive and we find its position in the red-black tree by comparing p with alive triangles (is it higher or lower), and then update the red-black tree by adding the new alive triangle. If we have k elements in the red-black tree, this operation is done in $O(\log k)$ (because $k \leq n$, we can say this operation is done in at most $O(\log n)$.
   - When a triangle ends, (it always ends with a point - let's call that point p), we just need to find the triangle in the red-black tree that has point p in the intersection with the current vertical sweep line and remove that triangle because it will no longer be alive. Comparisons of point p with the triangles in the red-black tree and then removing the dead triangle from the red-black tree can be done in $O(\log k)$ if there are k triangles in the red-black tree (because $k \leq n$, we can say this operation is done in at most $O(\log n)$.

- When we see a point p, we can check p with the alive triangles in the red-black tree to see if p falls in the vertical intervals that those triangles have in intersection with the current sweep line. If so, we know that p is inside a triangle. If not, we report p as a point that lies outside all triangles. This operation can be done in $O(\log k)$ if there are k triangles in the red-black tree (because $k \leq n$, we can say this operation is done in at most $O(\log n)$.

This way we handle all the points and check if they fall inside any triangle or not. We have $O(n)$ events and handle each of them in at most $O(\log n)$. Hence, the total running time of the algorithm is $O(n \log n)$.

Joseph Poremba

Problem 3.

Note: my proof and algorithm technically assume points on opposing staircases do not share the same y-coordinate, except for the top two points (i.e. steps on the two staircases are not at the same height). This is not too hard to fix, but is a little subtle and I did not want to introduce any technical errors.

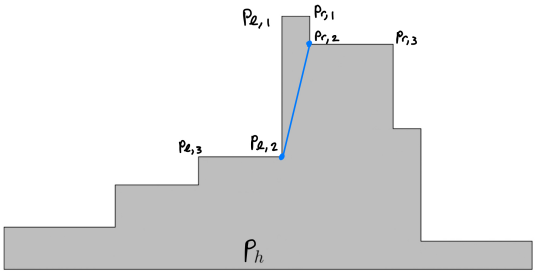
(a)

Some notation before we begin. Consider an orthogonal pyramid P. Let P_h be the horizontal edge whose length is the sum of all other horizontal edges. Without loss of generality we will say P_h is on the bottom of P. The "staircases" of P are two y-monotone chains (i.e. with respect to the vertical). The two monotone chains are connected by a topmost edge, which is horizontal. Let $P_{l,1}$ and $P_{r,1}$ be the left and right vertices on this edge. More generally we let $P_{l,i}$ and $P_{r,i}$ be the i-th vertices in the respective chains, starting from the top.

Note additionally that the two chains are also x-monotone (if one doubled-back horizontally, then P_h would not be long enough to meet the definition). This is clear from the "staircase" structure, but is worth stating explicitly.

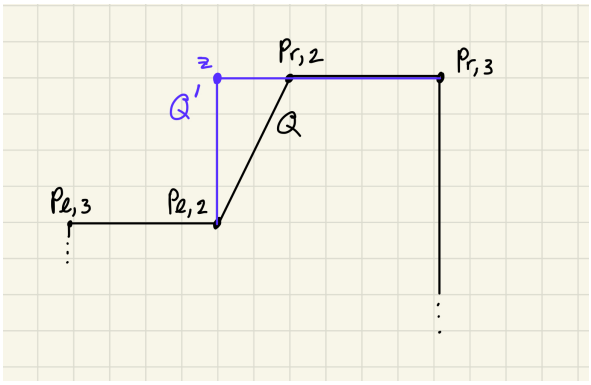
We prove the desired result by induction on n, the number of vertices. In fact we prove a slightly stronger result: there is a partitioning into convex quadrilaterals such that neither of the top two vertices $P_{l,1}$ nor $P_{r,1}$ is incident with a chord. The smallest orthogonal pyramid has $n = 4$, in which case the result trivially holds. So assume $n \geq 5$, and suppose the result holds for all smaller orthogonal pyramids.

Consider the line segment $e = \overline{P_{l,2}P_{r,2}}$.



First, we claim that e is a (legal) chord. The segment e does not intersect P_h, since h is a horizontal line that is no higher vertically than either of the ends of e. Similarly, the segment e avoids the topmost edge $\overline{P_{l,1}P_{r,1}}$. Furthermore, all edges of the left chain are to the left of $P_{l,2}$ (by x-monotonicity), and the segment e leaves $P_{l,2}$ to the right, so none of the edges of the left chain can intersect with e. A similar argument shows that e avoids all edges of the right chain. So it is a chord.

After adding this chord, the pieces obtained are the quadrilateral $P_{l,1}, P_{l,2}, P_{r,2}, P_{r,1}$, and another polygon Q. Note that the quadrilateral is convex since it contains two right angles (which, with a total of 360 degrees between all of its angles, makes it impossible to have a reflex angle.) It therefore suffices to show that Q has a partitioning into convex quadrilaterals. The polygon Q is almost an orthogonal pyramid, except it has one edge (namely our chord $P_{l,2}, P_{r,2}$) that is diagonal in place of one of the edges that should be vertical.



Without loss of generality, say $P_{l,2}$ is lower vertically than $P_{r,2}$. Note that the top two vertices of Q are $P_{r,2}$ and its horizontal neighbour $P_{r,3}$. Let z be the point with the same y-coordinate as $P_{r,2}$ and the same x-coordinate as $P_{l,2}$. Consider the polygon Q' obtained from Q by replacing $P_{r,2}$ with z (connected to both $P_{l,2}$ and $P_{r,3}$). Note that Q' is an orthogonal pyramid, and has fewer vertices than P. By induction, it has a partitioning into convex quadrilaterals, and furthermore that partitioning has no chords incident with z or $P_{r,3}$. By partitioning Q using the exact same chords as those for Q', we obtain a partitioning of Q into convex quadrilaterals. (Note that since no chord is incident with z or $P_{r,3}$, it must be the case that one of the quadrilaterals is $zP_{l,2}P_{r,4}P_{r,3}$. This quadrilateral stays convex when shifting z back to $P_{r,2}$.)

(Aside: Another way I can see doing this proof: rather than imposing the additional restriction prohibiting a chord incident with the top two vertices, we define an *almost orthogonal pyramid* to be like an orthogonal pyramid but allow the edge $\overline{P_{l,1}P_{l,2}}$ or the edge $\overline{P_{r,1}P_{r,2}}$ to be diagonal, and show the result holds for almost orthogonal pyramids.)

(b)

We essentially convert the above proof into an algorithm. Our algorithm takes as input the orthogonal pyramid and its top two vertices $P_{l,1}, P_{r,1}$.

`Partition`$(P, P_{l,1}, P_{r,1})$:

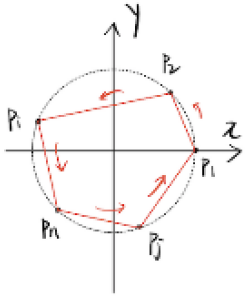
1. $\mathcal{C} \leftarrow \{\overline{P_{l,2}P_{r,2}}\}$ (the set of chords).
2. If $P_{l,2}$ has lower y-coordinate than $P_{r,2}$, let $i = l$ and $\bar{i} = r$. Otherwise let $i = r$ and $\bar{i} = l$.
3. Let z be the point with the same y-coordinate as $P_{\bar{i},2}$ and same x-coordinate as $P_{i,2}$.
4. Define polygon Q' by taking P, removing the vertices $P_{l,1}, P_{r,1}$, and $P_{\bar{i},2}$ (and all incident edges), adding the point z, and adding the edges $\overline{P_{i,2}z}$ and $\overline{zP_{i,3}}$.
5. $\mathcal{C} \leftarrow \mathcal{C} \cup$ `Partition`$(Q', z, P_{i,3})$.
6. Output $\mathcal{C}$.

Through a linear scan, it would take $O(n)$ time to find the initial $P_{l,1}, P_{r,1}$ to feed into `Partition`. Each recursive call operates on a polygon with two fewer points, and (assuming a reasonable data structure like a linked list) all the traversals and manipulations in one recursive step are constant time, so the runtime recurrence is $T(n) = T(n-2) + O(1)$. The runtime is thus $O(n)$.

4. **Pseudo code:**

$p_1 \leftarrow$ rightmost vertex (largest x-coordinate)
sort $p_2, ..., p_n$ by angle between edge $|p_1 p_i|$ and y-axis positive direction.
return p_1 and $sorted(p_2, ..., p_n)$

The running time is $O(n \log n)$ because there is a sorting step. The lower bound of the all algorithm is $O(n \log n)$ because we can take any algorithm for constructing simple polygon to construct a sorting algorithm: for an array of n numbers, normalize to range $\theta[0, 2\pi]$, map the point to 2D by $(\sin\theta, \cos\theta)$, set p_1 to be the vertex at $(1, 0)$, and run the algorithm of constructing a simple polygon. Therefore, lower bound of sorting gives the lower bound of constructing simple polygons.



Yibo Jiao