**Code Jam 2021 - Round 3**

# Analysis: Fence Design

This problem asks us to find a triangulation of the given set of points that uses two particular edges. Many of the arguments that follow are referenced in the linked article, so you can use it as an introduction.

The first step to solve this problem is to notice some properties of the finished job. Let $P$ be the input set of poles and $F$ be a maximum set of fences with endpoints in $P$ that do not intersect other than at endpoints.

1. The fences that are the edges of the convex hull of $P$ are in $F$.
Proof: By definition, the edges of the convex hull do not intersect with any other possible fence. Therefore, they can be added to any set of fences that do not contain them without generating any invalid intersections, so any maximum set contains them all.

2. The size of $F$ is at most $3\mathbf{N} - 3 - c$ where $c$ is the size of the convex hull of $P$.
Proof: Consider the graph where poles are nodes and fences in $F$ are the edges. The graph is planar, so the generalized form of Euler's formula applies. The formula states that $\mathbf{N} + A = K + |F|$, where $A$ is the number of internal areas (not counting the outside) and $K$ the number of connected components of the graph.

Notice that each edge on the convex hull is adjacent to exactly one internal area, and other edges are adjacent to at most two internal areas. Therefore, the sum of the number of sides of all internal areas is at least $3A$ and that is counting each convex hull edge once and each other edge at most twice, so $3A \leq c + 2(|F| - c) = 2|F| - c$, which means $A \leq (2|F| - c)/3$. Replacing that in Euler's formula we obtain $\mathbf{N} + (2|F| - c)/3 \geq K + |F|$. It follows that $3\mathbf{N} + 2|F| - c \geq 3K + 3|F|$, and then $3\mathbf{N} - c - 3K \geq |F|$. Since $K \geq 1$, we obtain $|F| \leq 3\mathbf{N} - 3 - c$.

3. The size of $F$ is at least $3N - 3 - c$.
Proof: By induction. The base case is when all points in $P$ are on the convex hull. For that case, consider a set of all edges in the convex hull of $P$ plus any triangulation of $P$. This set has exactly $2N - 3$ edges, which is equal to $3\mathbf{N} - 3 - c$ when $\mathbf{N} = c$, proving that there exists a set of at least that size without any invalid intersections.

If there exists a point $p$ not on the convex hull of $P$, consider an optimal set of edges for $P - p$, which by inductive hypotheses has size $3(\mathbf{N} - 1) - 3 - c$. Because $|F| = 3(\mathbf{N} - 1) - c - 3$, and $3(\mathbf{N} - 1) - c - 3K \geq |F|$ from (2), $K$, the number of connected components, must be exactly one. Similarly, every internal area must be a triangle, with all edges in the convex hull being adjacent to one of them and all edges not in the convex hull being adjacent to two of them. By definition, $p$ is not on the convex hull. By the fact that there are no collinear triples, $p$ is not in an existing edge. Thus, $p$ is strictly contained in one of these triangles. Therefore, we can add edges from $p$ to each of the $3$ vertices of the triangle that contains $p$ to get a solution for our problem of size $3(\mathbf{N} - 1) - 3 - c + 3 = 3\mathbf{N} - 3 - c$.

From (2) and (3) above we know the exact number of fences we need to build (given the size of the convex hull of the set of poles). Moreover, we know that such an answer contains the convex hull of $P$ and every internal area delimited by fences in the output is a triangle. We can use this to devise algorithms to generate optimal sets.

## Test Set 1

There are many possible solutions for Test Set 1. For example, the procedure in the proof of point (3) above shows how to solve the problem with no pre-placed fences. There are ad-hoc ways to get around the problems with pre-placed fences, but they take a lot of work, and there is something simpler.

The proofs above show that any maximal set of fences is also maximum (notice that we only used maximality in our reasoning). Therefore, we can simply add fences to a set as long as they do not intersect with any previously added fence. This algorithm can accommodate pre-placed fences quite easily: simply start with them. There are $O(\mathbf{N}^2)$ potential fences to consider, and for each one we need to check whether it intersects any of the fences we already have. Since the solution overall is of size $O(\mathbf{N})$ this means checking $O(\mathbf{N}^3)$ intersections. Checking a pair of line segments to see if they intersect can be done in constant time, which means this algorithm takes $O(\mathbf{N}^3)$ time overall.

## Test Set 2

As in Test Set 1, there are lots of algorithms that solve this problem without considering the pre-placed fences, but only some of them are easy enough to adapt to them. For example, the procedure from the proof of (3) can be implemented efficiently: if the order in which we process points is randomized and we keep the current triangles in a tree-like structure to perform the search for a triangle efficiently, we can get an expected $O(\mathbf{N} \log \mathbf{N})$ time complexity. This leads to an algorithm similar to the incremental algorithm to compute a Delauney triangulation.

Another option is to modify the Graham Scan algorithm to efficiently find the convex hull to keep not only a convex hull of the visited points, but also all the triangles for the points inside it.

Unfortunately, while the algorithms above can work with a lot of ad-hoc code to accommodate the pre-placed fences, they become really cumbersome. Below we present some better alternatives.

Let $x$ be the intersection of both lines that are the infinite extension of a pre-placed fence. Since the fences don't intersect, $x$ can occur on one of them, but not on both. Let us call any pre-placed fence that does not contain $x$ $f_1$, and the other fence $f_2$. By their definitions, all of $f_2$ is on the same side of the line that extends $f_1$. We can recognize which fence can be $f_1$ by comparing whether the orientation of the two endpoints of a potential $f_1$ and each endpoint of $f_2$ is the same (that is, checking whether both endpoints of the candidate $f_2$ lie on the same side of the line that goes through $f_1$).

### Sweep-line

We can solve the problem without pre-placed fences with a sweep-line that considers the points in order of x-coordinate and maintains a convex hull of all the already seen points, as in the monotone chain convex hull algorithm. When considering a new point $p$, we simply connect it to all points from the already-seen set that do not cause intersections from $p$. Notice that those points are a continuous range of the right side of the convex hull of the already-seen set. Therefore, we can find those points efficiently with ternary searches on its right side.

To accommodate pre-placed fences, we first rotate the plane to make $f_2$ vertical (possibly scaling everything to use only integers) and then run the sweep line algorithm only on points that are on the same side of the line that goes through $f_1$ as $f_2$ (including both endpoints of $f_2$ and neither endpoint of $f_1$). Since $f_2$ is now vertical, it will be included by the algorithm in the set when its second endpoint is processed. After that, we rotate everything again to make $f_1$ vertical, and start the algorithm from the set and convex hull we already have (the endpoints of $f_1$ will be the first two points that are processed in this second pass). As before, $f_1$ will be added naturally by the algorithm.

Notice the accommodation of the pre-placed fences only requires linear time work, so the overall algorithm, just as the version without pre-placed fences, requires $O(\mathbf{N} \log \mathbf{N})$ time overall.

## Divide and conquer

This divide and conquer algorithm also has a correlate to computing the convex hull. The idea is simple: divide the set of points with a line that goes through $2$ points, compute the result of each side (both of which include those $2$ points), and then combine.

Let us call the two recursive results $P$ and $Q$. The convex hulls of both are convex polygons with a shared side. Then, we keep two current poles $p$ and $q$. Initially, both $p$ and $q$ are on the same endpoint of the shared side. Both move away from that shared side: $p$ through consecutive vertices of $P$ and $q$ through consecutive vertices of $Q$. Initially we move them both together. After that, let $p_0$ and $q_0$ be the previous pole where $p$ and $q$ were, respectively, and $p_1$ and $q_1$ be the next value for each (that is, $p_0p$ and $pp_1$ are adjacent sides of the convex hull of $P$ and $q_0q$ and $qq_1$ are adjacent sides of the convex hull of $Q$). Then,

- If $p_1q$ does not intersect $p_0p$, we set $p = p_1$.
- If $q_1p$ does not intersect $q_0q$, we set $q = q_1$.
- Otherwise, we stop.

Each time we move one, we add the fence connecting the current $p$ and $q$ to the result. When we are done, we do the same starting from the other endpoint of the shared side.

To divide evenly, we can pick any point $x$, sort the other points by angle, and pick the median as $y$, dividing by the line that goes through $x$ and $y$. Alternatively, we can pick points randomly. On average, the split of points would be about even (as we did in the proposed solution for Juggle Struggle: Part 1).

The work done to combine takes linear time and the work done to split takes either $O(\mathbf{N} \log \mathbf{N})$ time for the sorting version or linear time for the randomized version. Using the Master theorem we can then see that using the randomized version, the overall algorithm takes $O(\mathbf{N} \log \mathbf{N})$ time, and using the sorting version, the overall running time is $O(\mathbf{N} \log^2 \mathbf{N})$.