Code Jam 2022 - World Finals

Analysis: Triangles

Test Set 1

In Test Set 1 there are so few points that we can just try every possible way to assign them to triangles. With a maximum of $\mathbf{N} = 12$ points, there are $12!/(3!^4 \cdot 4!) = 15400$ ways of getting 3 or 4 triangles, even less to get 1 or 2, and a single way to get 0, which means there are less than $4 \cdot 15400 + 1 = 61601$ things to try. For each one, we need to check that every triangle is valid (i.e., made up of non-collinear points) and that each pair of triangles meets the definition given in the statement. Since there are very few triangles, this requires quite a bit of code but not a lot of computation time.

Test Set 2

We solve this problem by constructively proving the following theorem: a set S of $3t$ points can be split to form t triangles that fulfill the conditions of the statement if and only if it does not contain a subset of $2t + 1$ collinear points.

The case where $t = 1$ is trivial. If $t > 1$ we consider 3 different cases. Let C be the largest subset of S made up of collinear points.

- If $|C| < 2t - 1$, we find a triangle that is separated from all other points by a line and then recursively solve an instance with less points. Since a triangle can use at most 2 points from C, this does not make the remaining set exceed the maximum number of collinear points allowed. One way of finding such a triangle is to get the 2 maximal points in the lexicographical order (i.e., ordering by X-coordinate and breaking ties by Y-coordinate) v and w, and then find the third points x_1 and x_2 that minimize and maximize, respectively, the angle vwx_i, breaking ties by the distance $|wx_i|$. Then, pick x_1 if the angle is less than π or x_2 otherwise (in that case, vwx_2 is guaranteed to be greater than π). Notice that this makes The line wx_i separate the three points from all others (there could be more points over that line, but w and x_i are the extreme points over it, so any triangles we can make from the remaining points will not interfere with this triangle).
- If $|C| \geq 2t - 1$ and $|C| \neq 3$, let $B, D \subseteq S$ be the subsets of points on each side of the line that goes through C. If neither is empty, assuming without loss of generality that $|B| \leq |D|$, we can match the lexicographically greatest $2|B|$ points of C, let us call that C' with B to recursively solve $B \cup C'$ and $D \cup (C \setminus C_1)$. Notice that there is a separation line between those two sets, so the recursive solutions do not interfere with each other. If one of B or C is empty we can take the two lexicographically greatest points in C and match it with one of the remaining points, as in the previous step, solving the rest recursively. If $|C| = 2$, this makes a single triangle. Otherwise, $|C| > 3$, so after removing two points from C and one point from $S \setminus C$, the requirements of the theorem applies to the remaining points.
- Otherwise, $|C| = 3$ and $t = 2$. If the convex hull of S has 3 vertices, we can use those for one triangle, and the other 3 for the other. If the convex hull of S has 5 or 6 vertices, it contains at least two from C, so we can use those 2 plus a single intermediate or adjacent point to form one triangle, and the rest for the other. If the convex hull of C has 4 points, there are a few cases to consider depending on where the other 2 points are located, but all are solvable. Implementation-wise, we can simply try all possible ways to match it, since there is a small amount anyway.

Because of the above, we can do the following: find a largest set of collinear points in the input C. If it is larger than $\lceil 2\mathbf{N}/3 \rceil$, ignore points from it to make it equal to that amount. After that, ignore points not in C to make the set of non-ignored points have size multiple of 3. We will match all those points into triangles using the above idea. If we keep the points sorted lexicographically, we can implement each step finding a new triangle in linear time, except for the fact that after each step of the first type we may need to recalculate C.

To speed up the calculation of largest subset of collinear points, we can use three techniques. The first two speed it up in complexity, but can be really slow in practice due to a combination of large constants and an accumulation of logarithmic factors depending on the exact implementation.

- Calculate all subsets of collinear points and keep them updated and on a priority queue to quickly identify the largest. There are $\mathbf{N}$ point removals and each one updates up to $\mathbf{N}-1$ sets, so there are $O(\mathbf{N}^2)$ updates overall. If the sets are linked lists and the priority queue is over an array (the sizes are limited to the range 1 through $\mathbf{N}$), this can be theoretically be done in $O(\mathbf{N}^2)$ overall. However, since an initial calculation of C for the entire input necessitates $O(\mathbf{N}^2 \log \mathbf{N})$ operations, it might be tempting to use less efficient but quicker to code structures.
- If $|C|$ is a lot smaller than the threshold needed to not be in the first case, we can simply take multiple steps of the first case without recalculating. It can be shown that doing this leads to only a logarithmic number of recalculations. Notice that this still makes the overall complexity $O(\mathbf{N}^2 \log^2 \mathbf{N})$ and it has large constants because the input size before each recalculation is not halved, but reduced by about $1/5$ in the worst case.

A simpler and a lot faster way is to not calculate C explicitly at all. Just keep doing the first case until you detect all remaining points are collinear (this happens when both vwx_1 and vwx_2 are planar angles). At that point, undo just enough of the latest made triangles to make that identified set of collinear points big enough to not be in the first case, and continue with the second and third cases. This makes the overall time complexity $O(\mathbf{N}^2)$ and, not having complicated structures or logarithms with small bases, it does not hide any large constants.