Code Jam 2019 - Round 3

Analysis: Napkin Folding

Test Set 1

For Test Set 1, we want to find exactly one folding segment such that when we fold the napkin across the segment, the regions on either side of the segment line up perfectly. Because the folding segment must split the napkin into two symmetric regions, we can show that each of the folding segment's endpoints either coincides with a vertex of the polygon defining the napkin, or is the midpoint of an edge of the polygon. If we tried to make a folding segment connecting any other points, the two parts of the split edge could not possibly line up perfectly. Thus, all we need to do is try every line segment connecting any pair of points that are vertices or midpoints of the polygon's edges. Then, for each potential folding segment, we check whether it is valid by making sure it is fully contained within the polygon and that the two regions it creates are symmetric across the folding line segment.

Checking for intersections between line segments can be done by using only integers. Reflecting points across a line or taking the midpoint of a side normally could produce points with non-integer coordinates. But we can scale up our points initially such that if the reflection were to produce a point with non-integer coordinates, it could not possibly be one of the valid endpoints for folding line segments. Of course, we can also choose to work with fractions.

With **N** points in our polygon, we have 2×**N** points to choose from for our folding segment's endpoints. Each potential folding segment can be checked in O(**N**) time. Because there are O($\mathbf{N}^2$) possible segments to check, this results in an O($\mathbf{N}^3$) overall time complexity.

Note that in order to check for symmetry across a folding segment, we cannot just check that the sets of vertices of the polygons defining each region are symmetrical. Rather, we must show that for every edge of the polygon of one region, there exists exactly one edge in the polygon defining the second region which is symmetric across the folding line segment. In other words, the order in which the points appear on each side matters.

Finally, we can note that if a given line is indeed an axis of symmetry of the polygon, then it is guaranteed that it doesn't intersect the polygon more than twice. This means that we don't need an explicit check in the code for the folding segment not to intersect the polygon outside its endpoints. A similar simplification can be applied to the solution of Test set 2, coming up next.

Test Set 2

Since we want to draw a neat folding pattern of **K**-1 non-intersecting line segments, we must partition our napkin into **K** regions of identical size and shape, all of which are symmetric with other regions sharing common line segments. Each of these **K** regions is a polygon with edges that are made up of folding line segments and/or edges or parts of edges from the polygon defining the napkin. It can be shown that at least two of these regions are only adjacent to one line segment in our folding pattern, with the other edges that define the region's polygon coming from the original polygon. Let's call these regions *corner regions*.

If we have a corner region, we can reflect that region across its one folding line segment to find the region that must be adjacent to it. If we keep reflecting these regions across the edges of their polygons, being careful not to create intersecting regions or leave the polygon defining the napkin, we can reconstruct the neat folding pattern. Thus, every neat folding pattern can be uniquely defined by just one folding line segment connecting two points on the border of the

polygon defining the napkin. That segment cuts off a corner region which can be successively reflected to give us the entire folding pattern.

We need to consider pairs of points on the napkin's border defining a folding line segment. For a given candidate folding line segment, we can successively reflect the corner region we form to get the full folding pattern. If we use more than **K**-1 reflections, or, if after we finish reflecting we don't end up creating the original polygon, we know that the chosen line segment does not give us a neat folding pattern.

Now all that remains is to select all pairs of points on the napkin's border. Clearly we cannot test every pair of points with rational coordinates, because there are infinitely many such points. Rather, we can show that the endpoints of the line segments in our neat folding pattern must be either vertices or points on the polygon's edges that are X/Y of the way between consecutive vertices, where $1 \le X \le Y-1$ and $2 \le Y \le \mathbf{K}$ (the proof for this is below). Therefore, we can create all of these candidate points and check every pair. With $\mathbf{K} \le 10$ and $\mathbf{N} \le 200$, there are at most 6400 candidate points for the vertices of the line segments in our neat folding pattern. This puts an upper bound of 6400^2 on the number of pairs that we might need to check. But, we can reduce this significantly by only considering pairs which create a corner region with an area equal to 1/**K** of the napkin's area. Every point can pair with at most 2 other points to create a line segment which is fully within the polygon and splits off a corner region with the proper area. Therefore, we only need to check these pairs.

We can check if a single line segment from a pair of points gives us a valid folding pattern in O(**N**) time if we are careful to stop once a reflection creates a point which is not on the border of one of our polygon's line segments. We can precompute all the valid points and use a hash table for this check. Because all of the points are of the form X/Y × p, where p is a point with integer coordinates and Y is between 2 and **K**, we can multiply everything by lcm(2, 3, ..., **K**) at the beginning and then work in integers, dividing and simplifying fractions only for the output.

To summarize, the algorithm requires these steps:

- 1. Compute all possible endpoints. ($O(\mathbf{N} \times \mathbf{K}^2)$)
- 2. Find candidate segments to create a corner region. ($O(\mathbf{N} \times \mathbf{K}^2)$).
- 3. For each candidate, fold it up to **K**-1 times, checking that everything is valid.

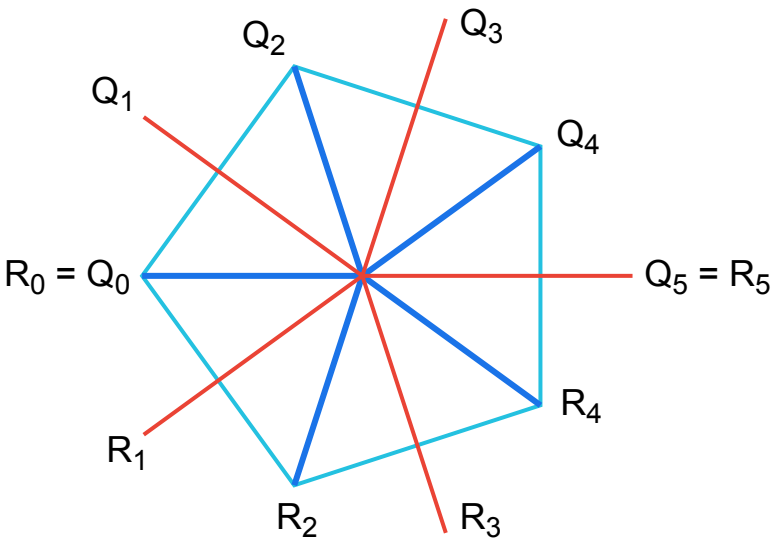
There are up to $O(\mathbf{N} \times \mathbf{K}^2)$ possible endpoints. If we fix one endpoint P and iterate the other, we can compute the area as we advance, finding the up to 2 segments that have an endpoint in P in $O(\mathbf{N} \times \mathbf{K}^2)$ time. So step 2 takes $O(\mathbf{N}^2 \times \mathbf{K}^4)$ time in total. For step 3, each unfolding requires reflecting the current region and finding new folding segments. All of that is linear in the size of the region, and the sum over the sizes of all regions is at most the total number of endpoints plus 2 × **K** shared ones, so this takes $O(\mathbf{N} \times \mathbf{K}^2)$ time overall. Putting it all together, the algorithm takes $O(\mathbf{N}^2 \times \mathbf{K}^4)$ time in total. It is possible to make it significantly more efficient than that, but the implementation is complicated enough as it is.

Appendix

To prove that all folding segments endpoints are X/Y of the way between consecutive vertices for $1 \le X \le Y-1$ and $2 \le Y \le \mathbf{K}$, we can prove that the number of folding segments that are incident on a non-vertex point of the input polygon is odd or zero. If that's true, let P be an endpoint and Q and R be the two vertices and/or folding segment endpoints that are closest to P towards each side over the same polygon edge E as P. Then, by reflection PQ and PR are equal. By repeating this over E we can see that E is divided into equal length sections by every point that is a folding segment endpoint.

Now we prove that the number of incident folding segments in a non-vertex point of the polygon is odd or zero. Assume that P is a point over an edge E of the polygon with I > 1 of incident folding segments. Let Q_i for i = 1, 2, ..., I be the non-P endpoints of each of those segments, in clockwise order. Let Q_0 be the reflection of Q_2 across PQ_1 and Q_{I+1} the reflection of Q_{I-1} across PQ_I. Notice both those points have to be on E. Now, the I+1 angles Q_iQ_{i+1} for i = 0, 1, ..., I are all equal. Let R_i be the reflection of Q_i across E. Now, because Q_i must reflect onto Q_{i+2}, the length of PQ_0, PQ_2, PQ_4, ..., PQ_I are all equal, and equal to the lengths of PR_2, PR_4, ..., PR_I. Since the angles between two consecutive segments of those are also all equal, $Q_0Q_2Q_4...Q_IR_IR_{I-2}...R_2$ is a regular polygon. All points have rational coordinates because they are either input points, endpoints of folding segments, or reflections calculated from other points with rational coordinates. It is known that the only regular polygon whose vertices have rational coordinates in 2 dimensions is the square, which means I / 2 + 1 + I / 2 = 4, which implies I = 3 is odd.

The following picture illustrates the above proof for the case I = 4. P is the point in the center, and line segments of the same color are guaranteed to be of the same length.



A consequence of this proof is that for any non-vertex point on the original polygon, it must be adjacent to exactly 0, 1 or 3 folding line segments.