FiveThirtyEight's June 26, 2020 Riddler

Emma Knight

June 29, 2020

This week's riddler is about drawing hexagons:

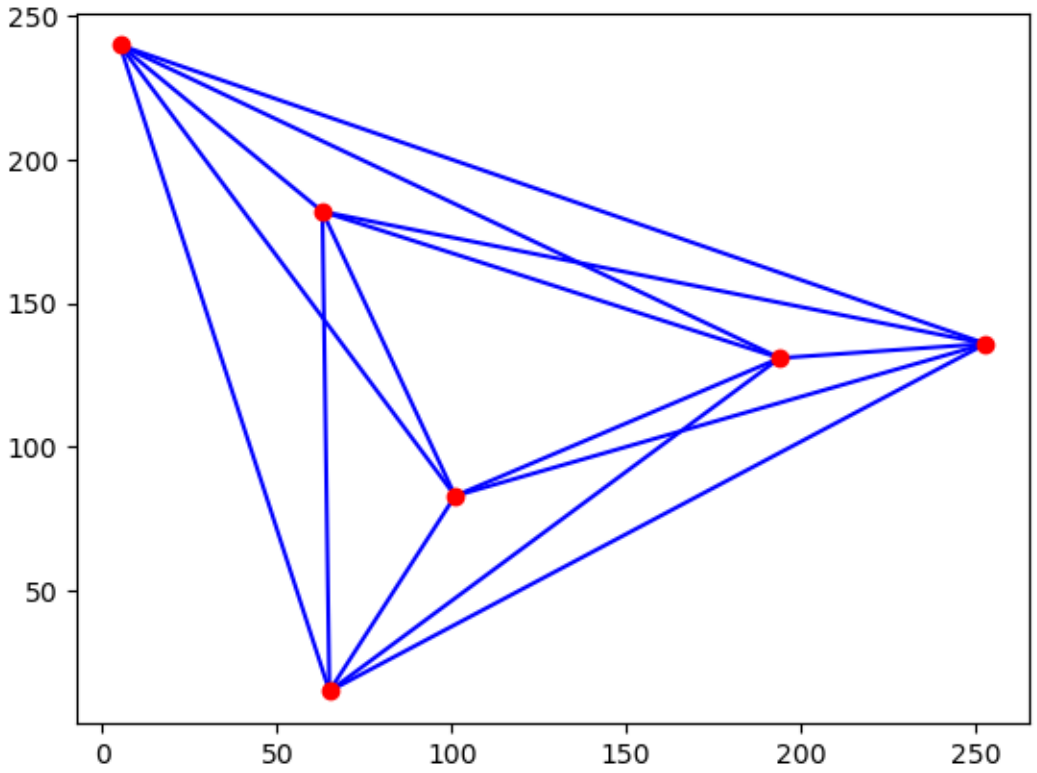
Question 1. *Polly Gawn loves to play "connect the dots." Today, she's playing a particularly challenging version of the game, which has six unlabeled dots on the page. She would like to connect them so that they form the vertices of a hexagon. To her surprise, she finds that there are many different hexagons she can draw, each with the same six vertices.*

*What is the greatest possible number of unique hexagons Polly can draw using six (*Extra Credit*: heptagons using seven) points?*

The point of this question (and the reason that the answer is not just $6!/(6 \cdot 2)$) is that sometimes, when you draw a cyclic graph on six verticies in the plane, two lines cross. The goal is to find the configuration with the most possible cyclic graphs that don't have a crossing.

The answers are 29 for the main question, and 92 for the extra credit.

The following statement appears sufficiently obvious that I'm going to leave it unjustified (it probably does merit having a proof though): the configuration of points with the most hexagons has the fewest crossings in the complete graph on six verticies drawn with lines. It is easy to prove that this number of crossings is at least three: if it was two, then there would be a vertex involved in both crossings. Deleting this vertex and all the edges adjacent to it would then lead to a planar embedding of K_5, a well-known impossibility. It is also fairly easy to draw K_6 in the plane with three crossings:



According to Oswin Aichholzer's website (link here) this is the unique way to draw K_6 in the plane with three crossings, so this must be the configuration of points. Thus, we need to count the number of hexagons that have at least one crossing. Label the crossings 1, 2, and 3 (which one is which doesn't matter because of symmetry). Let S_i be the set of hexagons that have crossing i; then we need to compute $|S_1 \cup S_2 \cup S_3|$. By the principle of inclusion-exclusion and symmetry, this number is $3|S_1| - 3|S_1 \cap S_2| + |S_1 \cap S_2 \cap S_3|$.

The size of S_1 is 12: pick one edge involved in crossing 1. Then in order to get the other edge, there are three times when you could insert that edge, there are two orientations of that edge you can use, and there are two ways to put the other two vertecies into the hexagon.

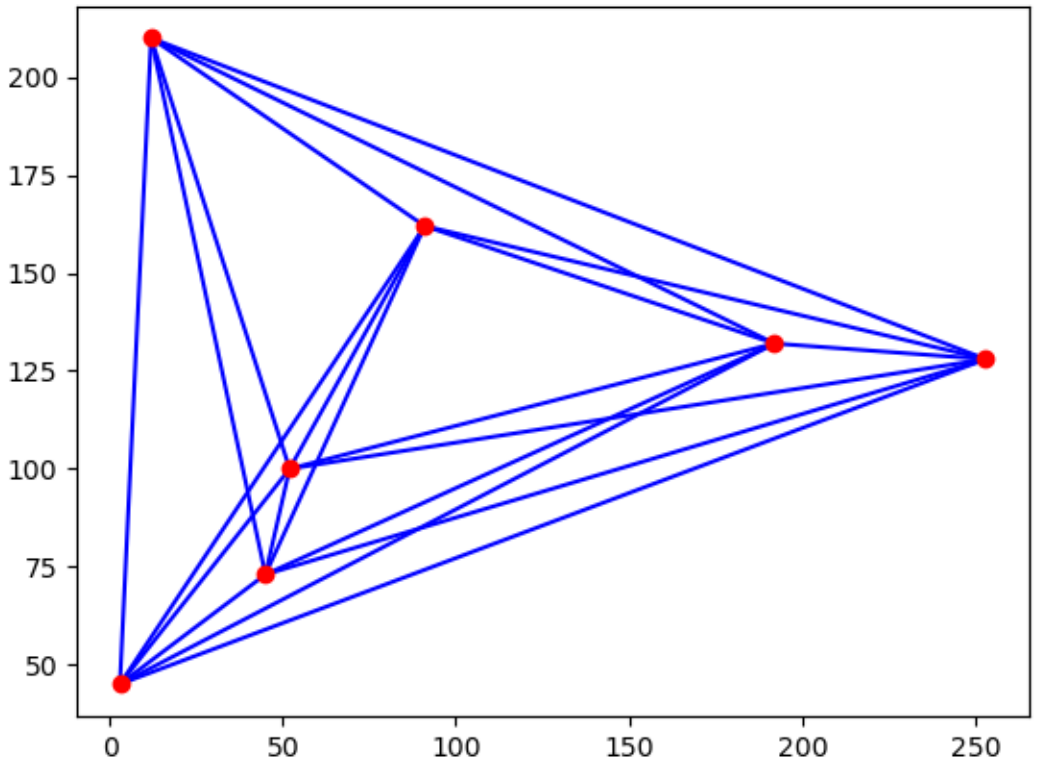
The size of $S_1 \cap S_2$ is 2: four of your edges are forced, and there are two remaining ways to add in the last two and get a hexagon instead of two triangles.

The size of $S_1 \cap S_2 \cap S_3$ is 1: all six edges are forced.

Thus, there are $36 - 6 + 1$ hexagons with a crossing, and hence 29 hexagons without one.

But what about for heptagons? I will grant the same assumption as before: you want to minimize the number of crossings when drawing your K_7 in order to find the most heptagons. According

to Oswin's site, there are three non-isomorphic embeddings of K_7 in the plane which attain the minimum number of crossings, which is 9. There is no way to count the number of heptagons that can be drawn on a given configuration of points by hand in finite time, so I will turn to a computer. According to the code below, the first configuration allows for 91 heptagons, the second one allows for 92, and the third one allows for 87 heptagons. Thus, the second configuration wins. The configuration looks like this:



And now, the code. First, I will give the code that both draws the K_7 and tells you which line segments intersect:

```
import matplotlib.pyplot as plt

##This is the code for drawing the K7s, and telling you which
##edges cross.  The three sets of points are below; to make the
##code run, just uncomment out the set of points you wish to
##use.  Again, thanks to Oswin Aichholzer for providing the
##sets of points (and they can be found at this link:
##http://www.ist.tugraz.at/staff/aichholzer/research/rp/triangulations/crossing/)

##xs = [183, 226, 186, 125, 127, 60, 29]
```

```
##ys = [255, 45, 64, 138, 158, 74, 59]

##xs = [3, 12, 91, 52, 45, 192, 253]
##ys = [45, 210, 162, 100, 73, 132, 128]

##xs = [134, 248, 180, 149, 143, 71, 8]
##ys = [235, 21, 60, 149, 108, 98, 72]

##This code is used to check if the line segment connecting
##p0 to p1 and the line segment connecting q0 to q1 intersect.
##Because Oswin gives point sets that have all points in general
##position (no three points lie on the same line) I don't need to
##handle any edge cases.

def direction(p, q, r):
    val = (q[1]-p[1])*(r[0]-q[0]) - (q[0]-p[0])*(r[1]-q[1])
    if val < 0:
        return -1
    if val > 0:
        return 1

def intersect(p0, p1, q0, q1):
    d1 = direction(p0, p1, q0)
    d2 = direction(p0, p1, q1)
    d3 = direction(q0, q1, p0)
    d4 = direction(q0, q1, p1)
    if (d1 != d2) and (d3 != d4):
        return True
    return False

##This loop finds all pairs of line segments on the graph that
##intersect, and prints out which ones do.  As a convienent
##sanity check, for all three sets of points, there are exactly
##nine pairs of line segments that intersect.

for i in range(7):
    for j in range(i+1, 7):
        for k in range(j+1, 7):
            for l in range(k+1, 7):
                pi = [xs[i], ys[i]]
                pj = [xs[j], ys[j]]
                pk = [xs[k], ys[k]]
                pl = [xs[l], ys[l]]
                if intersect(pi, pj, pk, pl):
                    print([[i+1, j+1], [k+1, l+1]])
                if intersect(pi, pk, pj, pl):
                    print([[i+1, k+1], [j+1, l+1]])
                if intersect(pi, pl, pj, pk):
```

```
                    print([[i+1, l+1], [j+1, k+1]])

##And now, to draw the graph.  This is standard matplotlib stuff.

for i in range(7):
    for j in range(i+1, 7):
        plt.plot([xs[i], xs[j]], [ys[i], ys[j]], 'b-')

plt.plot(xs, ys, 'ro')

plt.show()
```

Next, I will give the code that determines how many heptagons can be drawn from a given configuration:

```
import itertools

##This code tells you how many polygons can be drawn for
##a given set of points.  This code is actually entirely
##combinatorial: instead of checking to see if line
##segments intersect or anything like that, it just
##assumes that you have labeled the verticies of your
##graph and know which line segments intersect already.
##For the purposes of this code, a vertex is just a number,
##and an edge is just an implicitly unordered pair of numbers.

##These are some nice functions.  EdgeList takes in a list
##of verticies and produces the list of edges connecting
##each vertex to the next one on the list.  Contains takes
##two edges and a list of edges, and checks to see if the
##list of edges contains both of those edges.
##GeneratePolygons takes in an integer n and produces all
##n-gons on Kn.  Notice that GeneratePolygons will actually
##double count each polygon as it produces one polygon per
##orientation you can go around it.  It, however, always does
##start and end at vertex 1.  Purge takes two edges and a list
##of polygons,  and returns a list of all the polygons in the
##list that don't contain the two edges specified.

def EdgeList(verticies):
    output = []
    for i in range(len(verticies)-1):
        output.append([verticies[i], verticies[i+1]])
    return(output)

def Contains(edge1, edge2, edges):
    true1 = False
```

```
    true2 = False
    for edge in edges:
        if (edge[0] == edge1[0]) and (edge[1] == edge1[1]):
            true1 = True
        if (edge[1] == edge1[0]) and (edge[0] == edge1[1]):
            true1 = True
        if (edge[0] == edge2[0]) and (edge[1] == edge2[1]):
            true2 = True
        if (edge[1] == edge2[0]) and (edge[0] == edge2[1]):
            true2 = True
    return(true1 and true2)

def GeneratePolygons(n):
    output = []
    for l in list(itertools.permutations(range(2, n+1))):
        verticies = [1] + list(l) + [1]
        output.append(EdgeList(verticies))
    return(output)

def Purge(edge1, edge2, polygons):
    output = polygons.copy()
    for i in range(len(polygons)-1, -1, -1):
        if Contains(edge1, edge2, output[i]):
            del output[i]
    return(output)

##This is the main loop.  The three lists of edges
##correspond to the three embeddings of K7 constructed
##in the previous code.  In order to get the number of
##heptagons that you can draw on a specific K7, uncomment
##out the corresponding list of edges.

output = GeneratePolygons(7)

##l = [[[1, 3], [2, 4]],
##     [[1, 3], [2, 5]],
##     [[1, 4], [2, 5]],
##     [[1, 4], [3, 5]],
##     [[1, 6], [4, 7]],
##     [[1, 6], [5, 7]],
##     [[2, 4], [3, 5]],
##     [[2, 6], [3, 7]],
##     [[4, 7], [5, 6]]]

##l = [[[1, 3], [2, 4]],
##     [[1, 3], [2, 5]],
##     [[1, 4], [2, 5]],
##     [[1, 6], [4, 7]],
```

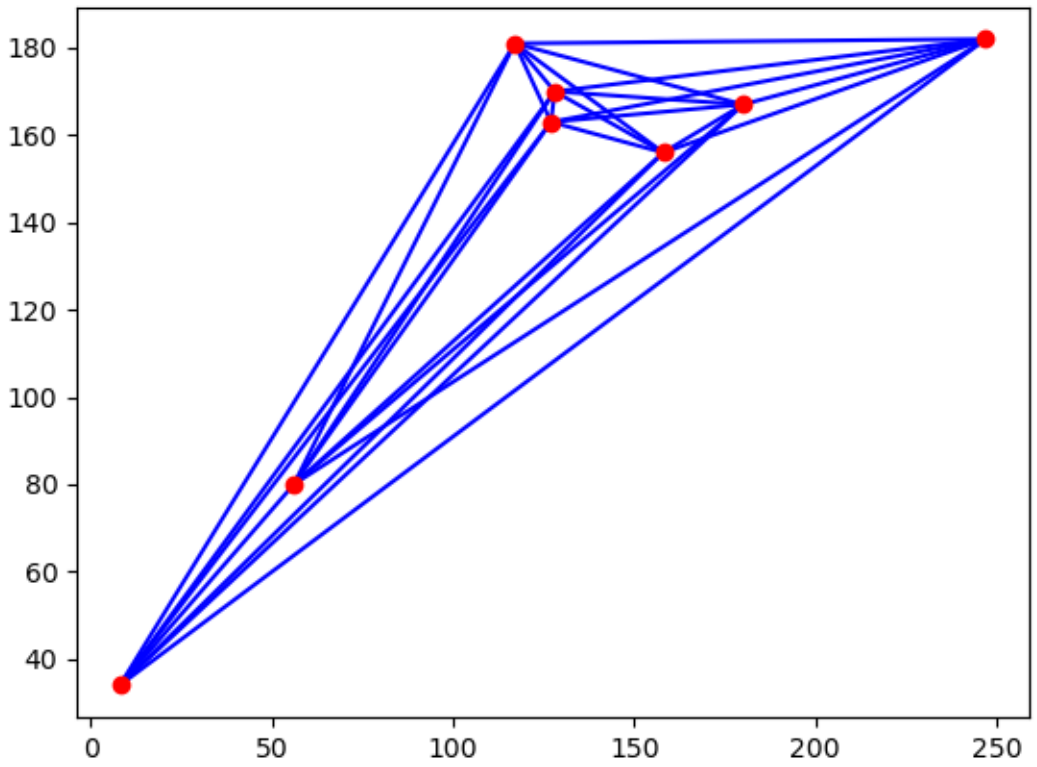
```
##     [[1, 6], [5, 7]],
##     [[2, 6], [3, 7]],
##     [[3, 5], [4, 6]],
##     [[3, 5], [4, 7]],
##     [[4, 7], [5, 6]]]

##l = [[[1, 3], [2, 4]],
##     [[1, 3], [2, 5]],
##     [[1, 5], [4, 6]],
##     [[1, 5], [4, 7]],
##     [[1, 6], [4, 7]],
##     [[2, 5], [3, 4]],
##     [[2, 6], [3, 7]],
##     [[2, 6], [5, 7]],
##     [[3, 6], [5, 7]]]

for i in range(len(l)):
    output = Purge(l[i][0], l[i][1], output)

print(len(output)//2)
```

Adapting this code for octogons, I got that the best arrangement, which allowed for 339 octagons, was:



At this point, I decided that I didn't want to run into the exponential growth for the size of the symmetric group and threw in the towel.

However, one can come up with a good heuristic guess for the largest number n-gons that can be drawn on n points in the plane. Notice that, for each crossing in your embedding of K_n in the plane, there are $(n-3)! \cdot 2$ different n-cycles that contain that crossing (call the first edge in the crossing 12 and say that your n-gon starts with 12. Then there are $n-3$ places to put the other edge, 2 orientations for it, and $(n-4)!$ different orderings for the remaining verticies). Thus, the odds that a randomly chosen n-cycle doesn't contain this crossing is $\left(1 - \frac{4}{(n-1)(n-2)}\right)$. Letting c_n be the rectilinear crossing number for K_n, the naieve guess for the number of n-gons is $\left(1 - \frac{4}{(n-1)(n-2)}\right)^{c_n} \cdot \left(\frac{(n-1)!}{2}\right)$. This gives 30.72 hexagons, 99.31... heptagons, and 376.32... octagons, which aren't that far off of the true values of 29, 92, and 339.