Code Jam 2021 - Round 2

Analysis: Matrygons

Test Set 1

The statement of this problem requires calculating a function from integers to integers. After we select a polygon to use, the remainder of the work is similar to the overall: select more polygons. This almost screams: write a recursive solution!

One way of doing that is with a top-down approach: start by picking the largest polygon, and then pick the rest. For all polygons except the largest, there is the additional restriction of "fitting" into the previous polygon. That is, the number of sides of the new polygon must be a proper divisor of the number of sides of the last one.

We can code that into a simple function that takes a number of sides left t and the size of the last polygon used p: $f(t,p) = 1 + \max_{q:q|p} f(t-q,q)$ (one plus the maximum $f(t-q,q)$ over all qs that are divisors of p) adding $f(0,p) = 0$ as base case, and appropriate conditions that q is indeed a polygon. The answer is then $\max_p f(\mathbf{N},p)$. This recursion is fast enough for the small bounds of Test Set 1 even if we iterate all possible $3 \le q \le \min(p-1,t)$. Of course, we can avoid that and find the divisors faster by iterating only up to $\sqrt{p}$ and checking both q and p/q, but the optimization is not necessary for Test Set 1, and not sufficient for Test Set 2.

Test Set 2

While memoization is the first instinct to speed up a recursive function, the function described has too large of a domain for that. A bottom-up approach, on the other hand, is much more suitable.

Consider starting from the smallest polygon. This can be done by using a function that is described by a similar expression, but swapping which of p and q is a parameter and which is iterated over: $f(t,q) = 1 + \max_{p:q|p} f(t-p,p)$. This switch allows for a significant speed up: when checking all possible p we can simply jump by q, dividing the total size of the iteration by q. This is enough to pass Test Set 2 if implemented carefully, but it may be hard to be confident that it is.

We can do better and improve our confidence with the following observation: once we pick a polygon of size q, all future polygons will have sizes multiples of q. Therefore, we can divide the whole problem by q (and allow pseudo-polygons of size 2) by setting $f(t,q) = f(t/q,1)$. If we are careful about not allowing size 2 for the very first polygon, this leaves us having to calculate only $g(t) = f(t,1)$, which has a domain of memoizable size. If we add memoization, we have a faster solution, and more importantly, one for which we can pre-compute the entire recursive function and be sure about the running time irrespective of the input.