Gearing Up For Destruction

Challenge 2 - *12/26/21 - 1/1/21*

Problem

We are given a list of distinct positive integers that represent the locations of pegs. Each location can be thought of as a coordinate on a number line.

On each peg, we must place a gear. Each peg must get one gear, and all gears must touch their adjacent gears. The goal is to come up with a configuration of gears, if it exists, that spins the last gear *twice* as fast as the first.

Gears can have a radius of any real number, with no constraints other than the radius must be greater than or equal to 1.

For example, if we are given a list of peg locations:

```
[5, 30, 50]
```

Then we could find gears with radii 12, 14 and 6 that would drive the last gear twice as fast as the first. The challenge has us return a `list` representing the simplified fraction in the form `[numerator, denominator]`. So, we would return:

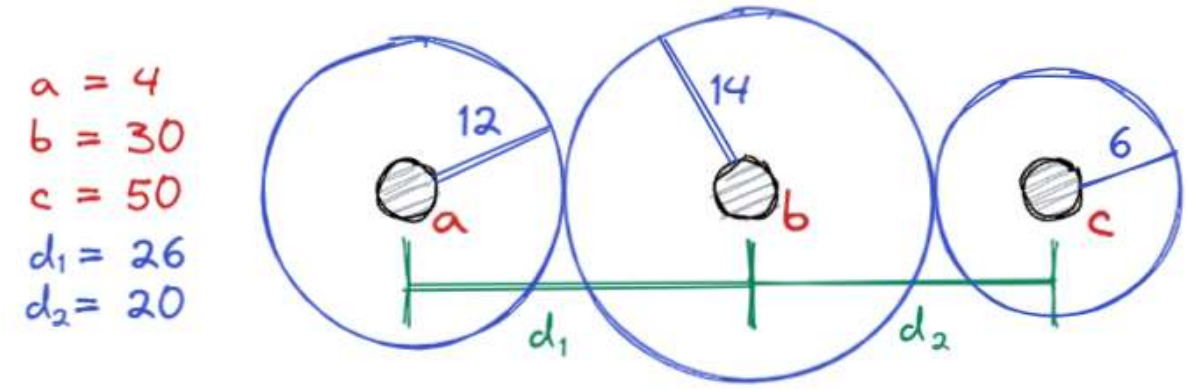
```
[12, 1]
```

Solution

Approach

Because the gears can have a radius of any real number, the amount of gear configurations is uncountably infinite. This eliminates any sort of searching algorithm, which was my naive approach.

Instead, we have to think about the problem as a system of equations, and solve for the radius of the first gear.



For the previous example with 3 gears, the system of equations is easy to solve. Firstly, we know that the distance between the pegs has to equal the sum of the gears' radii, being careful to double the middle gear's radius:

$$d_{total} = r_a + 2r_b + r_c$$

We also know that the radius of the last gear must be half the radius of the first gear, in order for it to spin twice as fast:

$$r_c = \frac{1}{2} r_a$$

Lastly, we know that the distance between pegs `a` and `b` must be equal to the radius of gear `a` plus the radius of gear `b`. Rearranged, this gives us a second equation to substitute:

$$d_1 = r_a + r_b$$

$$r_b = d_1 - r_a$$

Substituting all terms, we can solve for the radius of the first gear:

$$d_{total} = r_a + 2(d_1 - r_a) + \frac{1}{2} r_a$$

$$c - a = r_a + 2d_1 - 2r_a + \frac{1}{2} r_a$$

$$46 = 2d_1 - \frac{1}{2} r_a$$

$$r_a = -2(46 - 2d_1) = -2(46 - 2(30 - 4)) = -2(-6) = 12$$

In fact, we can come up with enough equations to substitute for however many gears we have, allowing us to use this same approach for `n` pegs.

Solving for n pegs

We can substitute equations for the distance between pegs for all gears except the first. This creates the following series:

$$r_0 + 2(d_0 - r_0) + 2(d_1 - (d_0 - r_0)) + 2(d_2 - (d_1 - (d_0 - r_0))) + \ldots + \frac{1}{2}r_0 = d_{total}$$

Each term simply adds a distance to subtract. Importantly, the simplified coefficient of r_0 can be determined without distributing each term. Since two adjacent terms will cancel out their r_0s, we know that for an odd number of pegs (and an odd number of terms) the coefficient will be -1/2, and 3/2 otherwise. This means we only have to solve for the summation of the distances.

So, assuming we can program a function to sum all the middle terms, we can solve for r_0:

$$c * r_o + f(pegs) = d_{total}$$

$$r_o = \frac{d_{total} - f(pegs)}{c}$$

Where `c` is `-1/2` or `3/2` depending on the parity of number of pegs, and `f(pegs)` is the function we write to add all the middle terms' constants.

Code

With this ironed out, the solution becomes pretty easy to transfer to code. Firstly, we should make a function that can solve any given term. It is very similar to implementing the Fibonacci sequence:

In [18]:
```
memo = {}
def nth_term(n, pegs):
    # Base case, if n is zero return the distance between
    # the first and second peg.
    if n == 0:
        return pegs[1] - pegs[0]
    # Otherwise, return the distance between this peg and the next,
    # and recurse.
    if (n,pegs) not in memo:
        memo[(n,pegs)] = (pegs[n + 1] - pegs[n]) - nth_term(n - 1, pegs)
    return memo[(n,pegs)]
```

Now, we can code the final solution.

In [16]:
```
from fractions import Fraction

def solution(pegs):
    total_distance = pegs[-1] - pegs[0]
    # The coefficient of the first gear radius:
    c = Fraction(3, 2) if len(pegs) % 2 == 0 else Fraction(-1, 2)
    # Number of middle terms to solve:
    n = len(pegs) - 2

    all_terms = sum(nth_term(i, pegs) for i in range(n)) * 2

    gear_radius = Fraction((total_distance - all_terms), c)

    return [gear_radius.numerator, gear_radius.denominator] if gear_radius >= 1 else [-
```

Testing it with the above three-gear example, we get the expected result:

In [19]:
```
solution([4, 30, 50])
```

Out[19]: [12, 1]

And when testing it with the provided example of an incompatible peg configuration, we get the proper sentinel values:

In [20]:
```
solution([4, 17, 50])
```

Out[20]: [-1, -1]