Project Eueler 512

Kevin Black

March 17, 2018

1 Problem Statement

Let $\phi(n)$ be Euler's totient function.

Let $f(n) = \left(\sum_{i=1}^{n} \phi(n^i)\right) \mod (n+1)$.

Let $g(n) = \sum_{i=1}^{n} f(i)$.

$g(100) = 2007$.

Find $g(5 \times 10^8)$.

2 Simplification

$$\begin{aligned}
f(n) &= \phi(n) + \phi(n^2) + \phi(n^3) + ... + \phi(n^n) \mod (n+1) \\
&= \phi(1 \cdot n) + \phi(n \cdot n) + \phi(n^2 \cdot n) + ... + \phi(n^{n-1} \cdot n) \mod (n+1) \\
&= 1 \cdot \phi(n) + n \cdot \phi(n) + n^2 \cdot \phi(n) + ... + n^{n-1} \cdot \phi(n) \mod (n+1) \\
&= \phi(n)\left(1 + n + n^2 + n^3 + ... + n^{n-1}\right) \mod (n+1) \\
&= (\phi(n) \mod (n+1))\left((1 + n + n^2 + n^3 + ... + n^{n-1}) \mod (n+1)\right) \mod (n+1)
\end{aligned}$$

Ignoring the $\phi(n)$ for now, and taking the right-hand side of the multiplication:

$$\begin{aligned}
1 + n + n^2 + ... + n^{n-1} &\equiv (1 + n + n^2 + n^3 + ... + n^{n-1}) - (n+1) && \mod (n+1) \\
&\equiv n^2 + n^3 + ... + n^{n-1} && \mod (n+1) \\
&\equiv n^2(1 + n + n^2 + n^3 + ... + n^{n-3}) && \mod (n+1) \\
&\equiv \left(n^2 \mod (n+1)\right)\left((1 + n + n^2 + n^3 + ... + n^{n-3}) \mod (n+1)\right) && \mod (n+1) \\
&\equiv 1 \cdot \left((1 + n + n^2 + n^3 + ... + n^{n-3}) \mod (n+1)\right) && \mod (n+1) \\
&\equiv 1 + n + n^2 + n^3 + ... + n^{n-3} && \mod (n+1)
\end{aligned}$$

The second-to-last step, where the $(n^2 \mod (n+1)$ is eliminated, is due to the fact that $n^2 \equiv (-1)^2 \equiv 1 \mod (n+1)$.

This process of eliminating $(1+n)$ and factoring out n^2 can be repeated, shrinking the series until there are no more terms left. If n is even, then the series will eventually become $(1+n) \mod (n+1)$, and subtracting out the final $(n+1)$ gives 0. If n is odd, then the series will eventually become $(1+n+n^2) \mod (n+1)$, and subtracting out the final $(n+1)$ gives $n^2 \mod (n+1)$, which is 1. Therefore, the final result is:

$$1 + n + n^2 + n^3 + ... + n^{n-1} \mod (n+1) = \begin{cases} 0 & n \text{ is even} \\ 1 & n \text{ is odd} \end{cases}$$

Plugging that back into the original equation, it gives:

$$f(n) = \begin{cases} 0 & n \text{ is even} \\ \phi(n) & n \text{ is odd} \end{cases}$$

The "mod $(n+1)$" can be dropped because $\phi(n)$ will never be greater than $n-1$.

This all means that $g(n)$ can be simplified to $\phi(1) + 0 + \phi(3) + 0 + \phi(5) + 0 + ... + \phi(n)$, or the sum of all odd totients between 1 and n, inclusive. This is still not trivial to compute for $n = 5 \cdot 10^8$, which the code below does.

3 Code

```
def computeOddTotients(n):
    m = (n - 1) // 2 # length of list, since only storing odd values

    # phi[i] corresponds to phi(i * 2 + 1), since we only care about odd values
    # start by marking each phi[i] as the number it's supposed to represent
    phi = [i * 2 + 1 for i in range(m + 1)]

    for i in range(1, m + 1):
        p = i * 2 + 1 # the actual odd number that phi[i] represents
        if (phi[i] == p):
            # phi[i] still has its initial value, so p is prime

            print(p)
            phi[i] = p - 1 # phi of a prime number p is p-1

            # update phi values of all multiples of p
            for j in range(i + p, m + 1 , p):
                # add contribution of p to its multiple i by multiplying by (1 - 1/p)
                phi[j] = (phi[j]//p) * (p-1)

    return phi

print(sum(computeOddTotients(5 * (10**8))))
```

4 Rationale

The function `computeOddTotients(n)` returns a list of $\phi(i)$ for all odd i between 1 and n, inclusive. It does so using a similar method to the Sieve of Eratosthenes to find all of the prime factors of each i, and uses Eueler's product formula to compute the totient of i, which is:

$$\phi(i) = i \prod_{p|i} (1 - \frac{1}{p})$$

A list of length $5 \cdot 10^8$ is extremely large and was very slow with the amount of memory I had available. Fortunately, we only care about odd totients, so we only technically need half as many elements, which is far more manageable. I accomplish this by creating a list `phi` encoded such that `phi[i]` $= \phi(2i+1)$, covering odd numbers only. I initialize each element to the number it is supposed to represent, $2i+1$; this indicates that the number has not been "visited" yet.

The Sieve-of-Eratosthenes-like algorithm then iterates through the entire list `phi`. If `phi[i]` is still equal to the odd number it's supposed to represent $(2i+1)$, then it has not been "visited" yet, and $2i+1$ (called p from now on) is prime. That means `phi[i]` can be directly computed as $p-1$. The algorithm then "visits" all of the multiples of the prime p, and changes them, marking them as not prime.

For each j that is a multiple of p that the algorithm visits, p is a prime factor of j, and thus Eueler's product formula can be used with p and j. Each j starts out with the value j (itself), accounting for the term outside of the product in Eueler's product formula. Then, each time j is visited by a prime factor $p|j$, it is multiplied by $(1-\frac{1}{p})$. (Note: in the actual code, j is distributed over $(1-\frac{1}{p})$ in order to avoid floating point division). Once the algorithm completes, every j will have been visited by every $p|j$, meaning every term in Euler's product formula will have been accounted for, and `phi` will hold all of the correct totients.