Project Euler: Problem 72 Writeup

A reduced proper fraction can be represented as $\frac{n}{d}$, where $n, d \in \mathbb{N}$, $n > d$, and $\gcd(n, d) = 1$. How many elements are contained in the set of reduced proper fractions $\{\frac{n}{d}\}$ for $d \leq 1000000$?

Of course, the brute-force method is rather straightforward: loop through all possible values (n, d) and use the Euclidean algorithm to check for values where $\gcd(\mathrm{n}, \mathrm{d}) = 1$. This is, however, inefficient; looping through all possible values of (n, d) runs in $O(n^2)$ time, and the use of the Euclidean algorithm adds to the time complexity.

Given some value of d, the number of elements in the set that contains reduced proper fractions that takes on that value is $\varphi(d)$, Euler's totient function, which gives the number of integers n, where $0 < n < d$, that satisfies $\gcd(n, d) = 1$. This is analogous to the original problem. Hence, we can write that

$$\sum_{k=1}^{1000000} \varphi(k) \tag{1}$$

is the solution to the problem. This can be generalized to

$$\sum_{k=1}^{N} \varphi(k), \tag{2}$$

where N is the maximum denominator. This is a rather profound observation, but does not say much about computation, as, the summation of $\varphi(k)$ still follows the aforementioned brute-force method.

However, there is a rather helpful identity of the totient function:

$$\varphi(k) = k \prod_{p|k} (1 - \frac{1}{p}), \tag{3}$$

where p is any prime that divides k. So, rather than brute-forcing the possibilities, we can construct a sieve similar to that of the Sieve of Eratosthenes - a list of lists where the list at index i contains all primes p that divide i, and, using the identity given at (3), compute the sum of the totient function of all values $0 < n \leq N$. Hence, we can write the following in pseudocode:

```
maxNumber = 1000000                          ▷ This can be replaced by another integer
primeDivisors = []

for i = 0 → maxNumber do                     ▷ Initialize list of lists
    primeDiviers[i].append([])
end for

for i = 2 → maxNumber do
    if primeDivisors[i].length = 0 then      ▷ Any i that fulfils this is prime
        j = i
        while j ≤ maxNumber do               ▷ Add i to list for all indices that are multiples of i
            primeDivisors[j].append(i)
            j = j + i
        end while
    end if
end for

sum = 0

for i = 2 → maxNumber do
    phi = primeDivisors[i][0]                ▷ Apply identity at (3)
    for p in primeDivisors[i] do
        phi = phi * (1 - 1/p)
    end for
    sum = sum + phi                          ▷ Add to sum
end for
```

27:
28: **return** sum

The above algorithm efficiently calculates the number of elements contained in the set of reduced proper fractions $\frac{n}{d}$, for $d \leq 1000000$, and can easily be adapted to work for any $d \leq N$. The code may look a little convoluted, but it is rather simple. The algorithm uses a filtering technique; for every prime, all multiples less than N is marked off by adding that prime to the list of divisors at index i, where i is any multiple of the given prime. Building up from 2, the next empty list is guaranteed to have a prime index, for all multiples of the primes preceding it have been marked off.

Of course, this is not the only way of solving the given problem. Another commonly used technique is related to Farey sequences.