Problem 72 - Counting Fractions

Gautam Manohar

22 July 2018

This document originally appeared as a blog post on my website. Find it at `gautammanohar.com/euler/72`.

1 Problem Statement

The fraction $\frac{a}{b}$ is called a reduced proper fraction if a and b are positive integers with $a < b$ and $\gcd(a, b) = 1$. If we list the reduced proper fractions for $b \leq 8$ in increasing order, we get

$$\frac{1}{8}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{3}{8}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{5}{8}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{7}{8}$$

How many elements are contained in the set of reduced proper fractions with denominator at most N?

2 My Algorithm

In order for a fraction to be in reduced form, the numerator and denominator must be coprime. And so there are $\varphi(n)$ fractions with denominator n. This is true for all n except for 1, which does not contribute any fractions, because we only consider the elements of $\mathbb{Q} \cap (0, 1)$. In total, there are

$$\sum_{n=1}^{N} \varphi(n) - 1 \tag{1}$$

fractions.

We use Euler's product formula for the totient function:

$$\varphi(n) = \prod_{p|n} \left(1 - \frac{1}{p}\right). \tag{2}$$

We can calculate this by using successive subtractions, starting from n. For example, the primes that divide 15 are 3 and 5. This gives $15 - \frac{15}{3} = 10 \to 10 - \frac{10}{5} = 8 = \phi(n)$.

We initialize an array `phi` of n `0`-entries. This accounts for $\varphi(1)$, which we do not wish to include in our final count. Then, we use a method similar to the Sieve of Eratosthenes. If `phi[n] = 0`, we have not yet dealt with n and its multiples. Because n is prime, we set `phi[n] = n - 1`. For all other multiples $kn < N$, we account for n being a prime factor by setting `phi[k*n] -= phi[k*n]//n`. Then, we sum the entries of `phi`, and add 1.

To give our final answer, we sum `phi[n]` for all $n < N$. Doing this once for each test case would have time complexity $O(Tn \log \log n)$. There are too many test cases for this, so we create a prefix sum array.

And so our solution has time complexity $O(n \log \log n + T)$, where T is the number of test cases.