Problem 800: Hybrid Integers

Rushil Surti

2022-06-10

1 Definition

We are given the definition of a hybrid integer as $H(p, q) = p^q q^p$, where $p \neq q$ and p and q are prime. We are also given the function $C(n)$ which is equal to the number of hybrid integers less than or equal to n.

2 Task

Find $C(800800^{800800})$.

3 Examples

$C(800) = 2$ because of the hybrid integers $2^3 3^2 = 8*9 = 72$ and $2^5 5^2 = 32*25 = 800$.

$C(800^{800}) = 10790$

4 Observations

The first observation we can make is that, because of the form of hybrid integers, the order of numbers does not matter. Because $H(2, 3) = H(3, 2)$, we do not have to check both. If we were to try iterating over a list of primes, this observation simplifies the process a bit. Instead of:

```
for p in prime_range(2, ...):
    for q in prime_range(2, ...):
        ...
```

We would now have:

```
for p in prime_range(2, ...):
    for q in prime_range(p + 1, ...):
        ...
```

Which is certainly faster.

The next observation is relating to the integer in the task. 800800^{800800} is a *really* big number. Checking with Haskell, the number is actually 4727543 digits long. Unless we can simplify this number, any type of iterating or brute force will not be possible.

The next observation we can make is that this problem involves a lot of exponents. It would be great to simplify all of these to a single base that would make comparisons much more trivial. If the reader has their mathematical cogs turning right now, they will probably agree with me saying that this is the time to use logarithms.

5 Doing the Math

How do we write numbers with exponents from one base to another?

This is the central question that this tackles. Thankfully, it is quite easy and only involves a little bit of algebra. Consider an example number like 2^3. We want to convert this to a base of e. The base does not necessarily have to be e, but ln is a common logarithm, and what is most important is that the bases are uniform.

$$\begin{aligned} 2^3 &= e^x \\ \ln 2^3 &= x \\ 3 \ln 2 &= x \\ 2^3 &= e^{3 \ln 2} \end{aligned} \tag{1}$$

Great! Getting out and dusting off our calculator (or our trusty python repl), we can confirm that this is true. Now, we can further generalize this:

$$\begin{aligned} a^b &= e^x \\ \ln a^b &= x \\ b \ln a &= x \\ a^b &= e^{b \ln a} \end{aligned} \tag{2}$$

If we wanted to, for example, use base 2 for performance reasons, we could also generalize the base to get:

$$\begin{aligned} a^b &= d^x \\ \log_d a^b &= x \\ b \log_d a &= x \\ a^b &= d^{b \log_d a} \end{aligned} \tag{3}$$

This is quite useful and can be applied to two critical places in our problem. The first of these places is the H function. We can rewrite it as follows:

$$\begin{aligned} H(p,q) &= p^q q^p \\ H(p,q) &= e^{q \ln p} e^{p \ln q} \\ H(p,q) &= e^{q \ln p + p \ln q} \end{aligned} \tag{4}$$

The reason for doing this becomes much more apparent when we rewrite 800800^{800800} as well.

$$800800^{800800} = e^{800800 \ln 800800} \tag{5}$$

From the description, we know that a hybrid integer is counted by the C function if it is less than or equal to some n. We can write this as:

$$p^q q^p \leq 800800^{800800} \tag{6}$$

But, now having our simplified forms, we can write is as:

$$e^{q \ln p + p \ln q} \leq e^{800800 \ln 800800} \tag{7}$$

Now that we have the same base on both sides, we can take the logarithm to simplify the inequality to:

$$q \ln p + p \ln q \leq 800800 \ln 800800 \tag{8}$$

For convenience sake, I will call the left side of the inequality $H1(p,q)$ and the right side the maximum bound.

This equality is *by far* much easier to iterate over than the previous one. The right side is somewhere in the 10 millions, which is much better. Now all that we need is some code and a lot of prime numbers.

6 Coding

The next question is how many primes we should calculate. If we set the first parameter to 2, we get the lowest values for the equation. Any other values will give higher results and reach maximum bound. This means that we need to find a q such that $H1(2,q) > 800800 \ln 800800$. This q will then be the range that we will have to calculate the primes to. The following python code finds this for us relatively quickly:

```
from math import log, e
from itertools import count

H1 = lambda p, q: p * log(q, e) + q * log(p, e)
maximum_bound = 800800 * log(800800, e)
```

```
for i in count(3):
    if H1(2, i) >= maximum_bound:
       print(i)
       break
```

After a slight bit of waiting, we get the result of 15704508. It does not really matter that this number is not prime because all it tells us is how far out we have to calculate our prime numbers. After we have our prime numbers, the number should be relatively easy to calculate with an algorithm like such written in pseudocode:

```
let primes = [...]
let total = 0

let H1(p, q) = p * log(q) + q * log(p)
let maximum_bound = 800800 * log(800800)

outer_loop:
for (let i = 0; i < length(primes); i++) {
    for (let j = i + 1; j < length(primes); j++) {
        let value = H1(primes[i], primes[j])
        // If the smallest q for some p is still over the bound, all
        // of the following p's and q's will too. No need to continue.
        if (float_greater(value, maximum_bound) and j == i + 1) break outer_loop;
        // If the value is greater but there still might be a pair
        // further on that satisfies the inequality, move onto the next p
        if (float_greater(value, maximum_bound)) break;
        // Otherwise, the value is still below the bound, so we can
        // just add one to the total and continue
        total++;
    }
}

print(total)
```

And, that's basically it. My C++ algorithm, which could honestly probably be optimized further, completed this in around 15 seconds, which isn't *too bad* in terms of speed. But anyways, let us congratulate ourselves on making it through! Actually, for a problem in the 800 range, it was quite a bit more trivial than I expected.