THE BROKEN IDEA ABOUT DISARIUM NUMBERS

Given the following sequence of disarium numbers

1, 2, 3, 4, 5, 6, 7, 8, 9, 89, 135, 175, 518, 598, 1306, 1676, 2427, 2646798

After multiple attempts to identify a pattern, I noticed that for a given disarium number X, its rightmost digit raise to its position number is significantly closer to X interms of their digit count and the equality of their leftmost digits. The rest of the digits of X provide a very minimal contribution in making X disarium.

For example

$6^4 \rightarrow 1306$, $length(6^4) = length(1306)$

$5^3 \rightarrow 135$, $length(5^3) = length(135)$

$6^4 \rightarrow 1676$, $length(6^4) = length(1676)$

$5^3 \rightarrow 175$, $length(5^3) = length(175)$

$7^4 \rightarrow 2427$, $length(7^4) = length(2427)$

$8^7 \rightarrow 2646798$, $length(8^7) = length(2646798)$

$8^3 \rightarrow 518$, $length(8^3) = length(518)$

$8^3 \rightarrow 598$, $length(8^3) = length(598)$

$9^2 \rightarrow 89$, $length(9^2) = length(89)$

One could use this to create a theory about disarium numbers but later on you'll notice that it does not provide any relevant information to increase performance by reducing the number of iterations.

With this worthless information, we have the certitude (which can still be wrong for an unknown disarium number) that for example a disarium number like 518 is such that.

$$8^3 < 518 < 599$$

So a solution to enumerate disarium numbers of length L consist of the following steps

1. Find all the Ys of length L such that

   $Y = M^N$ with $N, M \in [1, 9] \cap \mathbb{N}$

2. Given Y, Look for disarium numbers in the interval $[Y, E]$ where E is the largest number of length L with its leftmost digit equal to that of Y that's, it is of the form $I99999...9$.

```
#!/usr/bin/env perl

use strict;
use warnings;
```

```
sub is_disarium ($) {
    my $f;
    my @digs = split //, $_[0];

    while ( my ($k, $v) = each @digs ) {
        $f += $v ** ( $k + 1 );
        last if $f > $_[0]
    }

    return $f == $_[0] ? 1 : 0;
}


sub find_disarium ($) {
    my $c = $_[0];
    my (%box, @r);

    %box = (
                1 => [ 1, 2, 4, 8, 3, 9, 5, 6, 7 ],
                2 => [ 16, 32, 64, 27, 81, 25, 36, 49 ],
                3 => [ 128, 256, 512, 243, 729, 125, 625, 216, 343 ],
                4 => [ 2187, 6561, 1024, 4096, 3125, 1296, 7776, 2401 ],
                5 => [ 19683, 16384, 65536, 15625, 78125, 46656, 16807, 32768, 59049 ],
                6 => [ 262144, 390625, 279936, 117649, 823543, 531441 ],
                7 => [ 1953125, 1679616, 5764801, 2097152, 4782969 ],
            );

    OUTER: {
        foreach (sort keys %box) {
            # Find all disarium numbers of length $_

            my @found = map {
                my $c = 0;
                my $end = s[(.)(?:.(?{$c++}))*][$1.'9'x$c]er;

                map { is_disarium $_ ? $_ : () } ($_ .. $end);
            } $box{$_}->@*;

            foreach my $f (@found) {
                push @r, $f unless grep { $_ == $f } @r;
                last OUTER if @r >= $c;
            }
        }
    }
```

```
    return @r;
}

$| = 1;

my @dis = find_disarium 18;
print "Disarium numbers: @dis\n";
```

I first thought this program was optimal but after comparing its performance with that of a program which uses a naive approach(that's iterating from 0 until it finds all the 20 disarium numbers), I noticed that I was completly wrong and this idea is seriously a broken one.

Statistics

Broken idea

Real	User	Sys
8.31	8.24	0.05
8.75	8.67	0.06
8.58	8.52	0.05

The naive but not that naive method

Real	User	Sys
9.21	6.86	0.01
6.86	6.84	0.01
6.82	6.80	0.01