Minimal Set for Powers of 2

Bassam Abdul-Baki

October 7, 2019

Definition

Let x and y represent a string of digits (in base 10). x is defined to be a subsequence ($\lhd$) of y (denoted by x $\lhd$ y) if zero or more digits can be deleted from y, in any order, to get x (i.e., 24 $\lhd$ 1234).

A classical theorem of formal language theory, known as Higman's lemma[1], states that every set of pairwise incomparable strings is finite.

Given any set of strings S, we define the minimal set M(S) as the set of minimal elements of S such that for each x ə S, y ə M(S), and x $\lhd$ y, then x = y. Since M(S) is obviously pairwise incomparable, M(S) is also finite.

In 2001, Jeffrey Shallit[1] derived and proved the minimal set for primes, and conjectured on the minimal set for powers of 2.

Conjecture

If S = POWERS-OF-2 = {1, 2, 4, 8, 16, 32, 64, …}, then M(S) = {1, 2, 4, 8, 65536, ?}.

Shallit conjectured that the minimal set consists only of the known five numbers. My analysis suggests that there should be at least one extremely large number.

Theorem

My original unpublished proof is from 2010[3][4]. I'll reproduce it here for completeness's sake. What's new here are the lemmas and the data that support the difficult or unprovable nature of this problem.

Let $x_0 = 5$ and $x_{i+1} = 5x_i - 4$, then $\forall\, i \geq 1,\ i \in \mathbb{N}$,

$$16^{5^{i-1}+\left\lfloor\frac{i+3}{4}\right\rfloor} \equiv 16^{\left\lfloor\frac{i+3}{4}\right\rfloor}\ (mod\ 10^i) \tag{1}$$

Proof

$$\begin{aligned}
x_0 &= 5 \\
x_1 &= 5^2 - 4 \\
x_2 &= 5^3 - 4 \times 5 - 4 \\
x_{i+1} &= 5x_i - 4 \\
&= 5^{i+2} - 4 \times \sum_{j=0}^{i} 5^j \\
&= 5^{i+2} - 5^{i+1} + 1 \\
&= 4 \times 5^{i+1} + 1 \\
&\equiv 1\ (mod\ 4) \\
2^{x_{i+1}} - 2 &= 2^{5x_i - 4} - 2 \\
&= 2 \times (2^{5x_i - 5} - 1) \\
&= 2 \times \left(2^{5(x_i - 1)} - 1\right) \\
&= 2 \times (2^{x_i - 1} - 1) \times \left(2^{4(x_i - 1)} + 2^{3(x_i - 1)} + 2^{2(x_i - 1)} + 2^{x_i - 1} + 1\right) \qquad (2) \\
2^{x_i - 1} &= 2^{4 \times 5^{i-1}} \\
&= 16^{5^{i-1}} \\
&\equiv 1\ (mod\ 5) \\
\Rightarrow 2^{n(x_i - 1)} &\equiv 1\ (mod\ 5)
\end{aligned}$$

$$\Rightarrow 2^{4(x_i - 1)} + 2^{3(x_i - 1)} + 2^{2(x_i - 1)} + 2^{x_i - 1} + 1 \equiv 0\ (mod\ 5) \tag{3}$$

$\therefore$ By (2) and (3), $2^{x_{i+1}} - 2 = 5k \times (2^{x_i} - 2)$ (4)

By the Fermat-Euler Theorem, $2^4 \equiv 1\ (mod\ 5)\ and\ 2^5 \equiv 2\ (mod\ 5)$. (5)

$\therefore$ By induction on (2) using (4) and (5),

$$\begin{aligned}
2^{x_{i+1}} - 2 &= 2 \times (2^{x_i - 1} - 1) \times \left(2^{4(x_i - 1)} + 2^{3(x_i - 1)} + 2^{2(x_i - 1)} + 2^{x_i - 1} + 1\right) \\
&\equiv 0\ \left(mod\ 5 \times (2^{x_i} - 2)\right) \\
&\equiv 0\ \left(mod\ 5^{i+2}\right).
\end{aligned}$$

$Since\ 2\ ||\ (2^{x_i}-2), and\ (2,5)\ =\ 1, then$

$$2^i \times 2^{4\times 5^i+1} \equiv 2^{i+1}\ (mod 10^{i+1})$$
$$2^{4\times 5^i+i+1} \equiv 2^{i+1}\ (mod 10^{i+1})$$

Let j ∈ ℕ, ∃ i + 1 + j ≡ 0 (mod 4) for min j ≥ 0.

i	j	i+1+j
0	3	4
1	2	4
2	1	4
3	0	4
4	3	8
5	2	8
6	1	8
7	0	8
i	-(i+1) (mod 4)	$4 \times \left\lfloor \frac{i+4}{4} \right\rfloor$

$$2^{4\times 5^i+i+1} \equiv 2^{i+1}\ (mod\ 10^{i+1})$$
$$2^{4\times 5^i+i+1+j} \equiv 2^{i+1+j}\ (mod\ 10^{i+1})$$
$$2^{4\times 5^i+4\times\left\lfloor\frac{i+4}{4}\right\rfloor} \equiv 2^{4\times\left\lfloor\frac{i+4}{4}\right\rfloor}\ (mod\ 10^{i+1})$$
$$16^{5^i+\left\lfloor\frac{i+4}{4}\right\rfloor} \equiv 16^{\left\lfloor\frac{i+4}{4}\right\rfloor}\ (mod\ 10^{i+1})$$

Replacing i with i-1 completes the proof of (1).

■

Potential Solutions

Legend

Modulus 10^i

Potential powers of 16 (mod 10^i)

Potential powers of 16 (mod 10^{i-1}, but not mod 10^i)

Additive Delta – Differences of powers of 16 between (mod 10^1) and (mod 10^{i-1}) (i.e., The first residue digit)

Powers of 16					Additive Delta / Powers of 16 (mod 10^i)					# of Potential Solutions / Differences of the Residues				
i / n	**$n+5^{i-2}$**	**$n+2\times5^{i-2}$**	**$n+3\times5^{i-2}$**	**$n+4\times5^{i-2}$**	R_0	R_1	R_2	R_3	R_4					
1										**1**	**0**	**0**	**0**	**0**
1					6									
2					**+ 4×10 (mod 10^2)**					**0**	**1**	**1**	**1**	**1**
1	2	3	4	5	16	56	96	36	76	1	5	9	3	7
3					**+ 2×10^2 (mod 10^3)**					**3**	**2**	**3**	**1**	**3**
2	7	12	17	22	256	456	656	856	056	2	4	6	8	0
3	8	13	18	23	096	296	496	696	896	0	2	4	6	8
4	9	14	19	24	536	736	936	136	336	5	7	9	1	3
5	10	15	20	25	576	776	976	176	376	5	7	9	1	3
4					**+ 6×10^3 (mod 10^4)**					**9**	**7**	**8**	**6**	**6**
3	28	53	78	103	4096	0096	6096	2096	8096	4	0	6	2	8
4	29	54	79	104	5536	1536	7536	3536	9536	5	1	7	3	9
5	30	55	80	105	8576	4576	0576	6576	2576	8	4	0	6	2
9	34	59	84	109	6736	2736	8736	4736	0736	6	2	8	4	0
10	35	60	85	110	7776	3776	9776	5776	1776	7	3	9	5	1
12	37	62	87	112	0656	6656	2656	8656	4656	0	6	2	8	4
14	39	64	89	114	7936	3936	9936	5936	1936	7	3	9	5	1
15	40	65	90	115	6976	2976	8976	4976	0976	6	2	8	4	0
18	43	68	93	118	3696	9696	5696	1696	7696	3	9	5	1	7
22	47	72	97	122	1056	7056	3056	9056	5056	1	7	3	9	5
24	49	74	99	124	0336	6336	2336	8336	4336	0	6	2	8	4
25	50	75	100	125	5376	1376	7376	3376	9376	5	1	7	3	9

Powers of 16					Additive Delta / Powers of 16 (mod 10^i)					# of Potential Solutions / Differences of the Residues				
i / n	$n+5^{i-2}$	$n+2\times5^{i-2}$	$n+3\times5^{i-2}$	$n+4\times5^{i-2}$	R_0	R_1	R_2	R_3	R_4					
5					**+ 8×10^4 (mod 10^5)**					**18**	**22**	**23**	**21**	**23**
4	129	254	379	504	65536	45536	25536	5536	85536	6	4	2	0	8
9	134	259	384	509	76736	56736	36736	16736	96736	7	5	3	1	9
10	135	260	385	510	27776	7776	87776	67776	47776	2	0	8	6	4
12	137	262	387	512	10656	90656	70656	50656	30656	1	9	7	5	3
14	139	264	389	514	27936	7936	87936	67936	47936	2	0	8	6	4
15	140	265	390	515	46976	26976	6976	86976	66976	4	2	0	8	6
18	143	268	393	518	13696	93696	73696	53696	33696	1	9	7	5	3
24	149	274	399	524	50336	30336	10336	90336	70336	5	3	1	9	7
25	150	275	400	525	5376	85376	65376	45376	25376	0	8	6	4	2
28	153	278	403	528	20096	96	80096	60096	40096	2	0	8	6	4
35	160	285	410	535	23776	3776	83776	63776	43776	2	0	8	6	4
37	162	287	412	537	86656	66656	46656	26656	6656	8	6	4	2	0
39	164	289	414	539	83936	63936	43936	23936	3936	8	6	4	2	0
43	168	293	418	543	29696	9696	89696	69696	49696	2	0	8	6	4
47	172	297	422	547	57056	37056	17056	97056	77056	5	3	1	9	7
49	174	299	424	549	6336	86336	66336	46336	26336	0	8	6	4	2
53	178	303	428	553	36096	16096	96096	76096	56096	3	1	9	7	5
54	179	304	429	554	77536	57536	37536	17536	97536	7	5	3	1	9
55	180	305	430	555	40576	20576	576	80576	60576	4	2	0	8	6
60	185	310	435	560	19776	99776	79776	59776	39776	1	9	7	5	3
64	189	314	439	564	39936	19936	99936	79936	59936	3	1	9	7	5
68	193	318	443	568	45696	25696	5696	85696	65696	4	2	0	8	6
72	197	322	447	572	33056	13056	93056	73056	53056	3	1	9	7	5
75	200	325	450	575	97376	77376	57376	37376	17376	9	7	5	3	1
79	204	329	454	579	33536	13536	93536	73536	53536	3	1	9	7	5
80	205	330	455	580	36576	16576	96576	76576	56576	3	1	9	7	5
85	210	335	460	585	15776	95776	75776	55776	35776	1	9	7	5	3
89	214	339	464	589	95936	75936	55936	35936	15936	9	7	5	3	1
97	222	347	472	597	9056	89056	69056	49056	29056	0	8	6	4	2
100	225	350	475	600	93376	73376	53376	33376	13376	9	7	5	3	1
104	229	354	479	604	89536	69536	49536	29536	9536	8	6	4	2	0
109	234	359	484	609	736	80736	60736	40736	20736	0	8	6	4	2
115	240	365	490	615	30976	10976	90976	70976	50976	3	1	9	7	5
118	243	368	493	618	77696	57696	37696	17696	97696	7	5	3	1	9
122	247	372	497	622	85056	65056	45056	25056	5056	8	6	4	2	0
125	250	375	500	625	89376	69376	49376	29376	9376	8	6	4	2	0
6					**+ 4×10^5 (mod 10^6)**					**69**	**63**	**64**	**62**	**62**
7					**+ 2×10^6 (mod 10^7)**					**182**	**191**	**194**	**188**	**196**
8					**+ 6×10^7 (mod 10^8)**					**545**	**545**	**574**	**581**	**570**
9					**+ 8×10^8 (mod 10^9)**					**1,654**	**1,649**	**1,632**	**1,633**	**1,714**

Lemmas / Patterns

The following lemmas are used to prove the patterns in the Potential Solutions section.

Lemma 1

All powers of 16 when multiplied by $\left(16^{5^{i-2}}-1\right)$ are congruent (mod 10^i). Use one less than the bottom group number in the R_0 column of "Powers of 16 (mod 10^i)" in the Potential Solutions section.

Proof

To show that $\forall\ n \geq m \geq \left\lfloor\frac{i+3}{4}\right\rfloor, i \geq 2, m, n \in \mathbb{N}$,

$$16^n \times \left(16^{5^{i-2}}-1\right) \equiv 16^m \times \left(16^{5^{i-2}}-1\right)\ (mod\ 10^i)$$
$$or\ 16^m \times (16^{n-m}-1) \times \left(16^{5^{i-2}}-1\right) \equiv 0\ (mod\ 10^i)$$

we only have to show that the two equations hold (mod 2^i) and (mod 5^i) since (2, 5) = 1.

For (mod 2^i):

Let $i = 4\times a + b$, $0 \leq b \leq 3$, then

$$m \geq \begin{cases} a, & i = 0\ (mod\ 4) \\ a+1, & i \neq 0\ (mod\ 4) \end{cases}$$

For i = 0 (mod 4), $16^m = 2^{4\times m} \geq 2^{4\times a} = 2^i$.

For i ≠ 0 (mod 4), $16^m = 2^{4\times m} \geq 2^{4\times(a+1)} > 2^i$.

$\therefore 2^i \mid 16^m$ or $16^m \equiv 0$ (mod 2^i). (6)

For (mod 5^i):

$$16 = 15 + 1 \equiv 1\ (mod\ 5)$$

Assume $\exists\ i, j \mid \forall\ j, 1 \leq j \leq i$,

$$16^{5^{j-1}} \equiv 1\ (mod\ 5^j)$$

Then by strong induction,

$$16^{5^i} = \left(16^{5^{i-1}}\right)^5$$

$$= (5^i k + 1)^5$$
$$\equiv 1 \ (mod\ 5^{i+1}) \quad (7)$$
$$\Rightarrow 16^{5^{i-2}} \equiv 1 \ (mod\ 5^{i-1})$$

$Since\ 5 \mid (16^{n-m} - 1), then\ (16^{n-m} - 1) \times \left(16^{5^{i-2}} - 1\right) \equiv 0 \ (mod\ 5^i).$

∴ By (6) and (8), $16^m \times (16^{n-m} - 1) \times \left(16^{5^{i-2}} - 1\right) \equiv 0 \ (mod\ 10^i), \forall\ n \geq m \geq \left\lfloor \frac{i+3}{4} \right\rfloor.$ (8)

Note: $16^n \times \left(16^{5^{i-2}} - 1\right) \not\equiv 0 \ (mod\ 10^i)$ by the main Theorem.

■

Example

∀ n ∈ ℕ, 1 ≤ n ≤ 25,

$$16^n \times \left(16^{5^{3-2}} - 1\right) = 1{,}048{,}575 \times 16^n \equiv 200 \ (mod\ 10^3)$$

Lemma 2

The differences of row-wise adjacent powers of 16 is constant modulus 10^i. See the orange digits in the "Powers of 16 (mod 10^i)" columns (R_i = residue) in the Potential Solutions section. This lemma is a rewording of Lemma 1.

Proof

From Lemma 1, ∀ m, n ≥ $\left\lfloor \frac{i+3}{4} \right\rfloor$, i ≥ 2, m, n ∈ ℕ,

$$16^n \times \left(16^{5^{i-2}} - 1\right) \equiv 16^m \times \left(16^{5^{i-2}} - 1\right) \ (mod\ 10^i)$$

But applying the Distributive Property gives us the difference of row-wise adjacent powers.

$$16^{n+5^{i-2}} - 16^n \equiv 16^{m+5^{i-2}} - 16^m \ (mod\ 10^i)$$

■

Example

∀ n ∈ ℕ, 1 ≤ n ≤ 25,

$$16^{n+5^{3-2}} - 16^n \equiv 200 \ (mod\ 10^3)$$

Lemma 3

The additive deltas are equal to $16^{5^{i-2}} \times \left(16^{5^{i-2}} - 1\right) \ (mod\ 10^i)$. Use one less than the bottom group number in the R_0 column of "Powers of 16 (mod 10^i)" in the Potential Solutions section. This lemma is a particular instance of Lemma 2.

Proof

∀ i ≥ 2, i ∈ ℕ,

$$\begin{aligned} 16^{2\times 5^{i-2}} &= \left(16^{5^{i-2}}\right)^2 \ (mod\ 10^i) \\ &= 16^{5^{i-2}} \times \left(16^{5^{i-2}} - 1\right) + 16^{5^{i-2}} \ (mod\ 10^i) \\ \Rightarrow 16^{2\times 5^{i-2}} - 16^{5^{i-2}} &= 16^{5^{i-2}} \times \left(16^{5^{i-2}} - 1\right) \ (mod\ 10^i) \end{aligned}$$

By Lemma 1 and Lemma 2, this completes the proof.

■

Example

$$\begin{array}{lcll} 16^{2+5^{3-2}} - 16^2 & = & 16^7 - 16^5 & \equiv \ 200 \ \ (mod\ 10^3) \\ 16^{5^{3-2}} \times \left(16^{5^{3-2}} - 1\right) & = & 16^5 \times (16^5 - 1) & \equiv \ 200 \ \ (mod\ 10^3) \end{array}$$

Lemma 4

The differences of row-wise adjacent powers in a given modulus group is an even non-zero multiple of 10^{i-1}. See the orange cells in the "Additive Delta" column in the Potential Solutions section.

Proof

From Lemma 2, ∀ n ≥ $\left\lfloor \frac{i+3}{4} \right\rfloor$, i ≥ 2, n ∈ ℕ,

$$16^{n+5^{i-2}} - 16^n \equiv constant\ (C)\ (mod\ 10^i)$$

$$\begin{aligned} \sum_{k=0}^{4} \left(16^{n+(k+1)\times 5^{i-2}} - 16^{n+k\times 5^{i-2}}\right) &= 16^{n+5^{i-1}} - 16^n \\ &\equiv 5C\ (mod\ 10^i) \end{aligned}$$

By the main Theorem,

$$\begin{aligned} 16^n \times \left(16^{5^{i-2}} - 1\right) &\not\equiv 0\ (mod\ 10^i) \\ 16^{n+5^{i-1}} - 16^n &\equiv 0\ (mod\ 10^i) \end{aligned}$$

∴ C ≡ 0 (mod $2\times 10^{i-1}$), but C ≢ 0 (mod 10^i).

■

Example

∀ n ∈ ℕ, 1 ≤ n ≤ 25,

$$16^{n+5^{3-2}} - 16^n \equiv 200 \ (mod\ 10^3)$$

$$\sum_{k=0}^{4} \left(16^{n+(k+1)\times 5^{i-2}} - 16^{n+k\times 5^{i-2}}\right) = 1000 \equiv 5C \ (mod\ 10^3)$$

$\therefore C \equiv 0 \ (\mathrm{mod}\ 2 \times 10^2)$.

Lemma 5

The unit digit of $\frac{1}{5^i} \times \left(16^{5^{i-1}} - 1\right)$ is equal to 3.

Proof

By (7), $16^{5^{i-1}} \equiv 1 \ (mod\ 5^i)$.

$$\therefore 16^{5^{i-1}} - 1 = 5^i \times k$$

Given

$$16 - 1 = 5 \times 3$$
$$16^5 - 1 = 5^2 \times 41{,}493$$

Assume $16^{5^{i-1}} - 1 = 5^i \times (10A + 3)$.

Then by induction,

$$\begin{aligned}
16^{5^i} - 1 &= \left(16^{5^{i-1}} - 1\right) \times \left(\sum_{j=0}^{4} 16^{j\times 5^{i-1}}\right) \\
&= 5^i \times (10A + 3) \times \left(\sum_{j=0}^{4} \left(16^{5^{i-1}}\right)^j\right) \\
&= 5^i \times (10A + 3) \times \left(\sum_{j=0}^{4} (10B + 6)^j\right) \\
&= 5^i \times (10A + 3) \\
&\qquad \times (100C + 4 \times 10B \times 6^3 + 6^4 + 3 \times 10B \times 6^2 + 6^3 + 2 \times 10B \times 6 + 6^2 + 10B + 6 + 1) \\
&= 5^i \times (10A + 3) \times (100C + 9{,}850B + 1555) \\
&= 5^i \times (10A + 3) \times (100C + 9{,}850B + 1550 + 5) \\
&= 5^i \times (10A + 3) \times (50D + 5) \\
&= 5^i \times (10A + 3) \times 5 \times (10D + 1) \\
&= 5^{i+1} \times (10A' + 3)
\end{aligned}$$

■

Example

i	$a = 16^{5^{i-1}} (mod\ 10^{15})$	$k = \frac{a-1}{5^i}$	k (mod 10)
1	16	3	3
2	1,048,576	41,943	3
3	401,496,703,205,376	3,211,973,625,643	3
4	053,328,527,589,376	85,325,644,143	3
5	932,619,489,509,376	298,438,236,643	3
6	360,370,299,109,376	23,063,699,143	3
7	281,524,347,109,376	3,603,511,643	3
8	447,294,587,109,376	1,145,074,143	3
9	276,145,787,109,376	141,386,643	3
10	420,401,787,109,376	43,049,143	3
11	141,681,787,109,376	2,901,643	3
12	748,081,787,109,376	3,064,143	3
13	780,081,787,109,376	639,043	3
14	940,081,787,109,376	154,023	3
15	740,081,787,109,376	24,251	3

Note: Although k is shown using "a (mod 10^{15})", the actual value of k, for the full value of a, is correct up to the digits in **green**. The rest were too large for any program to correctly calculate the actual values of a. 10^{15} was chosen since that is the smallest power divisible by both 3 and 5. However, at i = 15, the pattern fails and 10^{30} should probably be used instead.

X+Y	((16^(X+Y)-1)/5^max(k)) (mod 10)				
x \ Y	1	2	3	4	5
0 +	3	1	9	7	3
5 +	3	1	9	7	1
10 +	3	1	9	7	9
15 +	3	1	9	7	7
20 +	3	1	9	7	3
25 +	3	1	9	7	3
30 +	3	1	9	7	1
35 +	3	1	9	7	9
40 +	3	1	9	7	7
45 +	3	1	9	7	1
50 +	3	1	9	7	3

The general rule for this appears to be as follows:

$$\frac{16^{X+Y}-1}{5^k}\ (mod\ 10) \quad = \quad \begin{cases} \frac{X+Y}{5}, & if\ X+Y \equiv 0\ (mod\ 5) \\ 3, & if\ X+Y \equiv 1\ (mod\ 5) \\ 1, & if\ X+Y \equiv 2\ (mod\ 5) \\ 9, & if\ X+Y \equiv 3\ (mod\ 5) \\ 7, & if\ X+Y \equiv 4\ (mod\ 5) \end{cases} , where\ 5^k \parallel (16^{X+Y}-1)$$

From this general pattern, one can see that every power of 5 implies $X + Y \equiv 1\ (mod\ 5)$. However, this pattern shall not be proven here. This pattern is similar to https://oeis.org/A001511 and others like it.

Lemma 6

The differences of powers in a given modulus is cyclical across all moduli (i.e., add 4, 2, 6, or 8 in that order). The sequences generated are either odd or even with the following patterns. See the orange cells in the "Additive Delta / Powers of 16 (mod 10^i)" columns in the Potential Solutions section.

Additive	Odd	Even
4	1, 5, 9, 3, 7	0, 4, 8, 2, 6
2	1, 3, 5, 7, 9	0, 2, 4, 6, 8
6	1, 7, 3, 9, 5	0, 6, 2, 8, 4
8	1, 9, 7, 5, 3	0, 8, 6, 4, 2

Proof

Lemma 2 shows that the delta between two consecutive powers for a given modulus is constant. Thus, we only have to prove this for any two cyclical powers.

Let $n \geq \left\lfloor \frac{i+3+j}{4} \right\rfloor, i \geq 2, 1 \leq j \leq 4, n \in \mathbb{N}$.

We need to determine for which j the following inequality holds.

$$\left(16^{n+5^{i-2}} - 16^n\right) \left(mod\ 10^i\right) - \left(16^{n+5^{i-2+j}} - 16^n\right) \left(mod\ 10^{i+j}\right)$$
$$= \left(16^n \times \left(16^{5^{i-2}} - 1\right)\right) \left(mod\ 10^i\right) - \left(16^n \times \left(16^{5^{i-2+j}} - 1\right)\right) \left(mod\ 10^{i+j}\right)$$

By (6), $16^n \equiv 0 \pmod{2^i}$ for the left-hand side and $16^n \equiv 0 \pmod{2^{i+j}}$ for the right-hand side.

Also, it is trivial to see that

$$A \equiv B \left(mod\ 10^i\right)$$
$$\Rightarrow A \times 10^j \equiv B \times 10^j \left(mod\ 10^{i+j}\right)$$

By Lemma 5,

$$\left(16^{5^{i-2}} - 1\right) = 5^{i-1} \times (10k + 3)$$

This gives us

$$\left(16^n \times \left(16^{5^{i-2}} - 1\right)\right) \left(mod\ 10^i\right) = 2^{4n} \times 5^{i-1}(10A + 3)$$
$$\left(16^n \times \left(16^{5^{i-2+j}} - 1\right)\right) \left(mod\ 10^{i+j}\right) = 2^{4n} \times 5^{i-1+j}(10B + 3)$$

Multiplying the first equation by 10^j and subtracting the second equation, we get

$$\begin{aligned}
0 &\equiv 2^{4n} \times 5^{i-1+j}(10A+3) \times 2^j - 2^{4n} \times 5^{i-1+j}(10B+3) \left(mod\ 10^{i+j}\right) \\
&= 2^{4n} \times 5^{i-1+j} \times \left((10A+3) \times 2^j - (10B+3)\right) \left(mod\ 10^{i+j}\right) \\
&= 2^{4n} \times 5^{i-1+j} \times \left(10\left(2^j \times A - B\right) + 3\left(2^j - 1\right)\right) \left(mod\ 10^{i+j}\right) \\
&= \left(\left(2^{4n} \times 5^{i-1+j} \times 10 \times \left(2^j \times A - B\right)\right) + \left(2^{4n} \times 5^{i-1+j} \times 3\left(2^j - 1\right)\right)\right) \left(mod\ 10^{i+j}\right) \\
&= \left(\left(2^{4n+1} \times 5^{i+j} \times \left(2^j \times A - B\right)\right) + \left(2^{4n} \times 5^{i-1+j} \times 3\left(2^j - 1\right)\right)\right)
\end{aligned}$$

Since $4n + 1 > i + j$, then $10^{i+j} | \left(2^{4n+1} \times 5^{i+j}\right)$.

Therefore, we need only check that

$$\left(2^{4n} \times 5^{i-1+j} \times 3\left(2^j - 1\right)\right) \equiv 0 \left(mod\ 10^{i+j}\right)$$

Since $4n > i - 1 + j$, then $2^{i+j} | 2^{4n}$.

$\left(2^j - 1\right) \equiv 0\ (mod\ 5)$ iff $j \equiv 0\ (mod\ 4)$.

Therefore, by induction on the first four powers, the additive delta is cyclical for every fourth power with the values shown.

■

Example

$$\begin{aligned}
\left(16^{4+5^{2-2}} - 16^4\right)\left(mod\ 10^2\right) &\equiv 4 \times 10^{2-1}\ \left(mod\ 10^2\right) = 40 \\
\left(16^{4+5^{6-2}} - 16^4\right)\left(mod\ 10^6\right) &\equiv 4 \times 10^{6-1}\ \left(mod\ 10^6\right) = 400000 \\
\left(16^{4+5^{4i}} - 16^4\right)\ \left(mod\ 10^{4i+2}\right) &\equiv 4 \times 10^{4i+1}\ \left(mod\ 10^{4i+2}\right) = 4 \times 10^{4i+1}
\end{aligned}$$

Lemma 7

If $a \equiv 0 \left(mod\ 2^i\right)\ and\ a \equiv 1 \left(mod\ 5^i\right)$, then $a \equiv 16^{5^{i-1}} \left(mod\ 10^i\right)$. This lemma is an alternate proof of Lemma 1.

Proof

By the Fermat-Euler Theorem, $b^{\varphi(n)} \equiv 1\ (mod\ n)$, where (b, n) = 1, $n = \prod_{i=1}^{k} p_i^{a_i}$, and $\varphi(n) = \prod_{i=1}^{k} \left(p_i^{a_i - 1} \times (p_i - 1)\right)$.

Therefore,

$$\begin{aligned} b^{\varphi(5^i)} &= b^{4 \times 5^{i-1}} \\ &= (b^4)^{5^{i-1}} \\ &\equiv 1\ (mod\ 5^i) \end{aligned}$$

Since $a \equiv 0\ (mod\ 2^i)$ and $5^{i-1} \geq i, \forall i \in \mathbb{N}$, then letting b = 2 gives us $16^{5^{i-1}} \equiv 0 \left(mod\ 2^i\right)$.

Since (2, 5) = 1, then $a \equiv 16^{5^{i-1}}\ (mod\ 10^i)$.

To prove uniqueness, let $a_1 = 5^i k_1 + 1$ and $a_2 = 5^i k_2 + 1$, where $k_1 \neq k_2$ and $a_1 \equiv a_2 \equiv 0 \left(mod\ 2^i\right)$.

Then,

$$\begin{aligned} a_1 - a_2 &= 5^i (k_1 - k_2) \\ &\equiv 0 \left(mod\ 2^i\right) \end{aligned}$$

and since $5^i \equiv 1\ (mod\ 2)$,

$$k_1 \equiv k_2 \left(mod\ 2^i\right)$$

Let $k_1 = 2^i m_1 + b$ and $k_2 = 2^i m_2 + b$.

Then,

$$\begin{aligned} a_1 - a_2 &= 5^i \left(\left(2^i m_1 + b\right) - \left(2^i m_2 + b\right)\right) \\ &= 5^i \left(2^i (m_1 - m_2)\right) \\ &\equiv 0 \left(mod\ 2^i\right) \end{aligned}$$

and, thus,

$$\begin{aligned} a_1 - a_2 &= 10^i (m_1 - m_2) \\ &\equiv 0 \left(mod\ 10^i\right) \\ \Rightarrow a_1 &\equiv a_2 \left(mod\ 10^i\right) \end{aligned}$$

$\therefore a \equiv 16^{5^{i-1}} \left(mod\ 10^i\right)$ is the unique solution.

■

Example

i	$a = 16^{5^{i-1}} \pmod{10^i}$	$k = \frac{a-1}{5^i}$	a (mod 2^i)	a (mod 5^i)
1	6	1	0	1
2	76	3	0	1
3	376	3	0	1
4	9,376	15	0	1
5	09,376	3	0	1
6	109,376	7	0	1
7	7,109,376	91	0	1
8	87,109,376	223	0	1
9	787,109,376	403	0	1
10	1,787,109,376	183	0	1
11	81,787,109,376	1,675	0	1
12	081,787,109,376	335	0	1
13	0,081,787,109,376	67	0	1
14	40,081,787,109,376	6,567	0	1
15	740,081,787,109,376	24,251	0	1

Note: See the table in the Example of Lemma 5.

Programming

In order to find all potential powers, one only has to calculate the previous known powers' potential solutions and add the additive delta of that modulus to get the next four leading digits to determine the potential solutions of the next higher power. This reduces the number of calculations to one fifth and reduces the time to calculate higher powers exponentially. The problem with this approach is storing all the numbers. We do gain speed, but we lose on memory; unless we read and write to a file, then speed takes a dive.

Combinations

If the delta differences of the residues are odd (see the first column in the "Differences of the Residues" columns in the Potential Solutions section), we are guaranteed four new solutions (O_4) since the first digit will be a 3, 5, 7, or 9 and the remaining number sequence has passed the previous modulus test.

If the delta differences of the residues are even, we are guaranteed at most two new solutions (E_2) since the first digit will be a 0 or 6 and the remaining number sequence has passed the previous modulus test. However, if the remaining number contains a "5*5*3" in that order, then the 6 will also be thrown out and we only have one new solution (E_1).

Thus, $O_4 + E_2 + E_1$ = Previous Total Solutions and $4 \times O_4 + 2 \times E_2 + E_1$ = Current Total Solutions.

i	O_4	⇔	E_2	E_1	# of Potential Solutions
1	–	<	–	1	1
2	1	>	–	–	4
3	2	=	2	–	12
4	6	=	6	–	36
5	18	=	17	1	107
6	54	>	51	2	320
7	160	=	151	9	951
8	477	>	433	41	2,815
9	1,407	<	1,246	162	8,282
10	4,143	>	3,526	613	24,237
11	12,126	>	9,957	2,154	70,572
12	35,276	<	27,983	7,313	204,383
13	102,188	<	78,251	23,944	589,198
14	294,595	<	217,558	77,045	1,690,541
15	845,242	<	602,000	243,299	4,828,267
16	2,414,020	<	1,658,335	755,912	13,728,662
17	19,434,939	>	12,431,316	7,003,557	38,869,812
18					109,605,945
19					307,872,443
20					861,587,221
21					2,402,681,620

Minimum Powers of 16

Every time the minimum power of 16 increases, the actual power of 16 increases by at least one digit for each power increase [$n \times \log_{10}(16)$]. Thus, for a solution to exist, the rate in which the potential power grows has to be much slower than the delta changes in power. If we can see that the minimum power of 16 (e.g., 598,640) remains a potential solution for a much longer duration than the next minimum potential solution to replace it (i.e., 30,617,335), then that would show that we are closing in on the number of digits yet to be eliminated. As it stands, the number of digits added far exceeds the number of digits eliminated.

j	Min Power of 16	# of Digits	Delta Digits	Digits Remaining
1	1	2	–	1
2	2	3	1	1
3	3	4	1	1
4	4	5	1	1
5	9	11	6	6
6	24	29	18	23
7	24	29	0	22
8	72	87	58	79
9	72	87	0	78
10	72	87	0	77
11	72	87	0	76
12	72	87	0	75
13	615	741	654	728
14	615	741	0	727
15	679	818	77	803
16	679	818	0	802
17	679	818	0	801
18	679	818	0	800
19	2,600	3,131	2,313	3,112
20	13,693	16,489	13,358	16,469
21	13,693	16,489	0	16,468
22	13,693	16,489	0	16,467
23	13,693	16,489	0	16,466
24	14,268	17,181	692	17,157
25	14,268	17,181	0	17,156
26	598,640	720,835	703,654	720,809
27	598,640	720,835	0	720,808
28	598,640	720,835	0	720,807
29	598,640	720,835	0	720,806
30	598,640	720,835	0	720,805

i	Min Power of 16	# of Digits	Delta Digits	Digits Remaining
31	598,640	720,835	0	720,804
32	598,640	720,835	0	720,803
33	598,640	720,835	0	720,802
34	598,640	720,835	0	720,801
35	30,617,335	36,866,945	36,146,110	36,866,910
36	30,617,335	36,866,945	0	36,866,909
37	30,617,335	36,866,945	0	36,866,908
38	5,351,076,993	6,443,338,737	6,406,471,792	6,443,338,699
39	5,351,076,993	6,443,338,737	0	6,443,338,698
40	5,351,076,993	6,443,338,737	0	6,443,338,697
41	5,671,125,054	6,828,715,002	385,376,265	6,828,714,961
42	37,929,362,189	45,671,502,942	38,842,787,940	45,671,502,900
43	37,929,362,189	45,671,502,942	0	45,671,502,899
44	37,929,362,189	45,671,502,942	0	45,671,502,898
45	341,764,098,390	411,524,980,226	365,853,477,284	411,524,980,181
46	753,012,834,643	906,717,801,391	495,192,821,165	906,717,801,345
47	753,012,834,643	906,717,801,391	0	906,717,801,344
48	949,121,243,560	1,142,855,855,334	236,138,053,943	1,142,855,855,286

Conclusion

Since the pattern cannot terminate because an odd cycle will obviously add four new potential solutions and an even cycle will add either one or two solutions, it remains to be seen if there isn't a power of 16 that does not reduce to a power of two from the set of {1, 2, 4, 8, 65536}.

One thing to note is that the potential solutions' list moves away from both limit ends for the following reasons. At the lower end, the limit shifts by one every four numbers, but the eliminated previous potential solutions appear to drop at a faster rate. At the upper end, the limit sees a drop depending on which of the largest numbers from the previous list is still a potential solution when $4 \times 5^{i-2}$ is added to it (e.g., For i = 5, the largest three solutions are 618, 622, and 625. However, for i = 6, the largest remaining potential solution is $618 + 4 \times 5^4 = 3{,}118 < 3{,}125$.). This trend, however, does not reduce the number of solutions since the sequence of potential solutions is non-decreasing until another solution is found and eliminated.

If one or more solutions do exist, then those numbers would be exceptionally large. Once a first solution is found, using a regular expression of it to eliminate most other potential solutions will prove difficult since checking for longer combination patterns grows exponentially. A distributed program would therefore be better suited to check for these solutions. If a solution cannot be zeroed in on, then the problem may be unproveable.

■

Observation

Based on the Lemmas / Patterns and the Potential Solutions, there might be an easier way to calculate modular arithmetic powers.

For example, looking at $16^{28} \pmod{10^4} = 0096$,

$28 = 4 \times 7 = 2^2 \times 7 = 2^2 \times (1 + 2 \times 3)$ or

$28 = 1 + 27 = 1 + 3^3$

To calculate $16^{28} \pmod{10^4}$ using the first breakdown, we get:

$$16^{28}\ (mod\ 10^4) = 16^{2^2 \times (1+2\times 3)}\ (mod\ 10^4) = (16^2)^2 \times (((16^2)^2)^2)^3\ (mod\ 10^4),$$

which gives us 6 multiplication operations, assuming $(16^2)^2$ is not calculated twice.

To calculate $16^{28} \pmod{10^4}$ using the second breakdown, we get:

$$16^{28}\ (mod\ 10^4) = 16^{1+3^3}\ (mod\ 10^4) = 16 \times ((16^3)^3)^3\ (mod\ 10^4),$$

which gives us 7 multiplication operations.

However, if we calculate $16^3 \pmod{10^4}$ using 2 multiplication operations, we would only need to add the additive delta to get the correct value for each consecutive power of 10 after 3 (i.e., 28, 53, 78, and 103). This reduces the number of multiplication operations drastically for the higher powers.

Developing a general formula for any power, assuming one exists or hasn't already been developed, is outside the scope of this paper.

Solutions Summary

i	Min Limit for Power of 16	Max Limit for Power of 16	# of Potential Solutions	Min Power of 16	Max Power of 16
1	1	1	1	1	1
2	1	5	4	2	5
3	1	25	12	3	25
4	1	125	36	4	125
5	2	626	107	9	625
6	2	3,126	320	24	3,118
7	2	15,626	951	24	15,618
8	2	78,126	2,815	72	78,118
9	3	390,627	8,282	72	390,618
10	3	1,953,127	24,237	72	1,953,118
11	3	9,765,627	70,572	72	9,765,539
12	3	48,828,127	204,383	72	48,828,039
13	4	244,140,628	589,198	615	244,140,539
14	4	1,220,703,128	1,690,541	615	1,220,702,509
15	4	6,103,515,628	4,828,267	679	6,103,515,009
16	4	30,517,578,128	13,728,662	679	30,517,577,509
17	5	152,587,890,629	38,869,812	679	152,587,890,009
18	5	762,939,453,129	109,605,945	679	762,939,452,509
19	5	3,814,697,265,629	307,872,443	2,600	3,814,697,265,009
20	5	19,073,486,328,129	861,587,221	13,693	19,073,486,305,797
21	6	95,367,431,640,630	2,402,681,620	13,693	95,367,431,606,618
22	6	476,837,158,203,130		13,693	476,837,158,169,118
23	6	2,384,185,791,015,630		13,693	2,384,185,790,950,547
24	6	11,920,928,955,078,130		14,268	11,920,928,954,929,785
25	7	59,604,644,775,390,631		14,268	59,604,644,775,219,697
26	7	298,023,223,876,953,131		598,640	298,023,223,876,782,197

i	Min Limit for Power of 16	Max Limit for Power of 16	# of Potential Solutions	Min Power of 16	Max Power of 16
27	7	1,490,116,119,384,765,631		598,640	1,490,116,119,384,594,697
28	7	7,450,580,596,923,828,131		598,640	7,450,580,596,923,380,330
29	8	37,252,902,984,619,140,632		598,640	37,252,902,984,618,692,830
30	8	186,264,514,923,095,703,132		598,640	186,264,514,923,092,438,993
31	8	931,322,574,615,478,515,632		598,640	931,322,574,615,462,248,693
32	8	4,656,612,873,077,392,578,132		598,640	4,656,612,873,077,357,546,685
33	9	23,283,064,365,386,962,890,633		598,640	23,283,064,365,386,927,859,185
34	9	116,415,321,826,934,814,453,133		598,640	116,415,321,826,934,779,421,685
35	9	582,076,609,134,674,072,265,633		30,617,335	582,076,609,134,674,037,234,185
36	9	2,910,383,045,673,370,361,328,133		30,617,335	2,910,383,045,673,370,326,296,685
37	10	14,551,915,228,366,851,806,640,634		30,617,335	14,551,915,228,366,851,606,543,560
38	10	72,759,576,141,834,259,033,203,134		5,351,076,993	72,759,576,141,834,256,896,013,200
39	10	363,797,880,709,171,295,166,015,634		5,351,076,993	363,797,880,709,171,293,028,825,700
40	10	1,818,989,403,545,856,475,830,078,134		5,351,076,993	1,818,989,403,545,856,473,692,888,200
41	11	9,094,947,017,729,282,379,150,390,635		5,671,125,054	9,094,947,017,729,282,371,782,856,555
42	11	45,474,735,088,646,411,895,751,953,135		37,929,362,189	45,474,735,088,646,411,881,401,569,118
43	11	227,373,675,443,232,059,478,759,765,635		37,929,362,189	227,373,675,443,232,059,464,409,381,618
44	11	1,136,868,377,216,160,297,393,798,828,135		37,929,362,189	1,136,868,377,216,160,297,249,736,267,803
45	12	5,684,341,886,080,801,486,968,994,140,636		341,764,098,390	5,684,341,886,080,801,486,824,931,580,303
46	12	28,421,709,430,404,007,434,844,970,703,136		753,012,834,643	28,421,709,430,404,007,434,700,908,142,803
47	12	142,108,547,152,020,037,174,224,853,515,636		753,012,834,643	142,108,547,152,020,037,174,080,790,955,303
48	12	710,542,735,760,100,185,871,124,267,578,136		949,121,243,560	

Python Source Code

The Python language[5] was chosen for its ease of use and infinite-precision arithmetic. There are different methods of programming this. One approach is to store all the previous solution powers and use them to build the next iteration. This, however, would run out of memory very quickly and storage space soon after. A second approach would be to find only the first power for each iteration and make sure that we are always less than the congruent limit or the maximum computed one. This approach is slightly better. We still need to calculate the power for almost every modulus, but it does not require as much memory and resources. I used the latter approach.

MinimalSets.py

Description

This algorithm can run forever, albeit very slowly. The upper limit value is used to avoid checking all the way up to the maximum upper bound for each module power.

Code

```
import sys
import math
import re

def minimalPowersOf2():
    pattern = re.compile('3.*?5.*?5.*?6')
    # This pattern assumes that unless you're starting at a certain non-zero length, those lengths will be skipped.
    # It is particularly noticeable for the first small powers.
    MP2 = [
         0,  8,  7,  6,  5,  4,  3,  2,  1,  0, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,
         0, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  1,  0,
         3,  2,  1,  0,  0,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,
         0,  3,  2,  1,  0,  0,  8,  7,  6,  5,  4,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,
         0,  8,  7,  6,  5,  4,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,  2,  1,  0, 15, 14, 13, 12, 11, 10,
         9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  0,  1,  0,  1,  0,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,
         3,  2,  1,  0,  6,  5,  4,  3,  2,  1,  0,  1,  0,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  6,  5,
         4,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,
```

```
     0,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  3,  2,  1,  0, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,
     0,  3,  2,  1,  0, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  3,  2,  1,  0,  2,  1,
     0,  8,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,  0,  2,  1,  0,  2,  1,  0,  6,  5,  4,  3,  2,  1,
     0, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,
     3,  2,  1,  0,  0,  0,  4,  3,  2,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  2,  1,
     0,  3,  2,  1,  0,  0,  4,  3,  2,  1,  0,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,
     0,  8,  7,  6,  5,  4,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,  2,  1,  0, 10,  9,  8,  7,  6,  5,
     4,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,  1,  0,  1,  0,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,
     3,  2,  1,  0,  6,  5,  4,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  3,  2,  1,  0,  5,  4,
     3,  2,  1,  0,  6,  5,  4,  3,  2,  1,  0,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,
     0,  3,  2,  1,  0,  0,  4,  3,  2,  1,  0,  3,  2,  1,  0, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,
     0, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0, 18, 17, 16, 15, 14, 13, 12, 11, 10,
     9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,  0,  2,  1,  0,  2,  1,  0,  5,  4,  3,  2,  1,  0,
    12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  5,  4,
     3,  2,  1,  0,  0,  0,  4,  3,  2,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  6,  5,
     4,  3,  2,  1,  0,  0,  4,  3,  2,  1,  0, 18, 17, 16, 15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,
     4,  3,  2,  1,  0, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,  0,  3,  2,  1,  0,  2,  1
    ]
array_size = len(MP2)
modLength = 49
current_power = 1791001362435
upper_limit = 320999043869
current_index = current_power % array_size
solutions = 0
curr_solution = current_power >> 32
prev_solution = curr_solution
max_power = 5**(modLength - 1) + int((modLength + 3)//4)
while current_power < max_power - upper_limit:
    truncPowerOf16 = str(pow(16, current_power, 10**modLength))
    isMSP = False
    if '1' not in truncPowerOf16:
        if '2' not in truncPowerOf16:
            if '4' not in truncPowerOf16:
                if '8' not in truncPowerOf16:
                    match = re.search(pattern, truncPowerOf16[::-1])
                    if not match:
                        isMSP = True
    if isMSP:
```

```
            print(modLength, max_power, current_power, sep = " - ")
            f = open(f"{modLength}_MS.txt", "w")
            f.write(str(current_power))
            f.close()
            modLength += 1
            solutions += 1
#           print_solutions = True
            max_power = 5**(modLength - 1) + int((modLength + 3)//4) - upper_limit #- 1
#           if len(str(powerOf16)) < modLength:
#               current_index += 1
        else:
            current_index += 1
            current_power += MP2[current_index] + 1
            current_index = (current_index + MP2[current_index]) % array_size
            curr_solution = current_power >> 32
            if prev_solution != curr_solution:
                # Not an accurate check for percentage left since the upper bound keeps decreasing.
                # It's an issue of speed vs. accuracy.
                percentage_left = round(current_power/upper_limit, 2)
#               percentage_left = round((math.log(current_power, 5))/(modLength - 1), 2)
                print(modLength, solutions, current_power, percentage_left, sep = " - ")
                prev_solution = curr_solution
                f = open(f"{modLength}_MS.txt", "w")
                f.write(str(current_power))
                f.close()
    print("QED", modLength, sep = " - ")

minimalPowersOf2()
```

ReverseMinimalSets.py

Description

This algorithm determines the upper limit for each module power.

Code

```
import sys
import math
import re

def reverseMinimalPowersOf2():
    pattern = re.compile('3.*?5.*?5.*?6')
    MP2 = [
          0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10, -11, -12, -13, -14,   0,
          0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -20, -21,   0,  -1,   0,
         -1,  -2,  -3,   0,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,   0,  -1,  -2,
          0,  -1,  -2,  -3,   0,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,   0,  -1,  -2,
          0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,   0,  -1,  -2,  -3,  -4,  -5,   0,  -1,  -2,   0,  -1,  -2,  -3,  -4,  -5,  -6,
         -7,  -8,  -9, -10, -11, -12, -13, -14, -15,   0,   0,  -1,   0,  -1,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,   0,
         -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,  -6,   0,  -1,   0,  -1,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,   0,  -1,  -2,
         -3,  -4,  -5,  -6,   0,  -1,  -2,  -3,  -4,  -5,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10, -11, -12, -13, -14,
          0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10,
          0,  -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10, -11, -12, -13,   0,  -1,  -2,  -3,   0,  -1,  -2,
          0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,   0,  -1,  -2,   0,  -1,  -2,   0,  -1,  -2,   0,  -1,  -2,  -3,  -4,  -5,  -6,
          0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -20, -21, -22, -23,   0,
         -1,  -2,  -3,   0,   0,   0,  -1,  -2,  -3,  -4,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,   0,  -1,  -2,
          0,  -1,  -2,  -3,   0,   0,  -1,  -2,  -3,  -4,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,   0,  -1,  -2,
          0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,   0,  -1,  -2,  -3,  -4,  -5,   0,  -1,  -2,   0,  -1,  -2,  -3,  -4,  -5,  -6,
         -7,  -8,  -9, -10,   0,  -1,  -2,  -3,  -4,  -5,   0,  -1,   0,  -1,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,   0,
         -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,  -6,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,   0,  -1,  -2,  -3,   0,  -1,  -2,
         -3,  -4,  -5,   0,  -1,  -2,  -3,  -4,  -5,  -6,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,   0,  -1,  -2,
          0,  -1,  -2,  -3,   0,   0,  -1,  -2,  -3,  -4,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10,
          0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10, -11, -12, -13, -14,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9,
        -10, -11, -12, -13, -14, -15, -16, -17, -18,   0,  -1,  -2,   0,  -1,  -2,   0,  -1,  -2,   0,  -1,  -2,  -3,  -4,  -5,   0,
         -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10, -11, -12,   0,  -1,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,   0,  -1,  -2,
         -3,  -4,  -5,   0,   0,   0,  -1,  -2,  -3,  -4,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,   0,  -1,  -2,  -3,   0,  -1,  -2,
         -3,  -4,  -5,  -6,   0,   0,  -1,  -2,  -3,  -4,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10, -11, -12, -13, -14,
        -15, -16, -17, -18,   0,  -1,  -2,  -3,  -4,  -5,  -6,  -7,  -8,  -9, -10,   0,  -1,  -2,   0,  -1,  -2,  -3,   0,  -1,  -2
```

```
        ]
    array_size = len(MP2)
    modLength = 48
    lower_limit = 1791001362435
    upper_limit = 320999043869
    max_power = 5**(modLength - 1) + int((modLength + 3)//4)
    current_power = 5**(modLength - 1) + int((modLength + 3)//4) - upper_limit
    current_index = current_power % array_size
    solutions = 0
    curr_solution = current_power >> 32
    prev_solution = curr_solution
    while current_power > lower_limit:
        truncPowerOf16 = str(pow(16, current_power, 10**modLength))
        isMSP = False
        if '1' not in truncPowerOf16:
            if '2' not in truncPowerOf16:
                if '4' not in truncPowerOf16:
                    if '8' not in truncPowerOf16:
                        match = re.search(pattern, truncPowerOf16[::-1])
                        if not match:
                            isMSP = True
        if isMSP:
            print(modLength, current_power, sep = " - ")
            f = open(f"{modLength}_RMS.txt", "w")
            f.write(str(upper_limit))
            f.close()
            modLength += 1
            solutions += 1
            #upper_limit = max_power - current_power
            max_power = 5**(modLength - 1) + int((modLength + 3)//4)
            current_power = max_power - upper_limit
            current_index = current_power % array_size
            increment_power = MP2[current_index]
            current_power += increment_power
            upper_limit -= increment_power
            current_index = (current_index + MP2[current_index]) % array_size
#           if len(str(powerOf16)) < modLength:
#               current_index += 1
```

```
        else:
            current_index -= 1
            increment_power = MP2[current_index] - 1
            current_power += increment_power
            upper_limit -= increment_power
            current_index = (current_index + MP2[current_index]) % array_size
            curr_solution = current_power >> 32
            if prev_solution != curr_solution:
                # Not an accurate check for percentage left since the upper bound keeps decreasing.
                # It's an issue of speed vs. accuracy.
                percentage_left = round(lower_limit/current_power, 2)
#               percentage_left = round((math.log(upper_limit, 5))/(modLength - 1), 2)
                print(modLength, solutions, upper_limit, percentage_left, sep = " - ")
                prev_solution = curr_solution
                f = open(f"{modLength}_RMS.txt", "w")
                f.write(str(upper_limit))
                f.close()
    print("QED", modLength, sep = " - ")

reverseMinimalPowersOf2()
```

MinimalSetsCount.py

Description

This algorithm determines the number of potential solutions and the divisions between odds and evens.

Code

```
import sys
import math
import re

def minimalPowersOf2(modLength):
    pattern = re.compile('3.*?5.*?5.*?6')
    pattern2 = re.compile('3.*?5.*?5')
    # This pattern assumes that unless you're starting at a certain non-zero length, those lengths will be skipped.
    # It is particularly noticeable for the first small powers.
    MP2 = [
     0,  8,  7,  6,  5,  4,  3,  2,  1,  0, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,
     0, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  1,  0,
     3,  2,  1,  0,  0,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,
     0,  3,  2,  1,  0,  0,  8,  7,  6,  5,  4,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,
     0,  8,  7,  6,  5,  4,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,  2,  1,  0, 15, 14, 13, 12, 11, 10,
     9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  0,  1,  0,  1,  0,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,
     3,  2,  1,  0,  6,  5,  4,  3,  2,  1,  0,  1,  0,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  6,  5,
     4,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,
     0,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  3,  2,  1,  0, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,
     0,  3,  2,  1,  0, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  3,  2,  1,  0,  2,  1,
     0,  8,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,  0,  2,  1,  0,  2,  1,  0,  6,  5,  4,  3,  2,  1,
     0, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,
     3,  2,  1,  0,  0,  0,  4,  3,  2,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  2,  1,
     0,  3,  2,  1,  0,  0,  4,  3,  2,  1,  0,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,
     0,  8,  7,  6,  5,  4,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,  2,  1,  0, 10,  9,  8,  7,  6,  5,
     4,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,  1,  0,  1,  0,  3,  2,  1,  0,  5,  4,  3,  2,  1,  0,
     3,  2,  1,  0,  6,  5,  4,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  3,  2,  1,  0,  5,  4,
     3,  2,  1,  0,  6,  5,  4,  3,  2,  1,  0,  3,  2,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,
     0,  3,  2,  1,  0,  0,  4,  3,  2,  1,  0,  3,  2,  1,  0, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,
```

```
     0, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0, 18, 17, 16, 15, 14, 13, 12, 11, 10,
     9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,  0,  2,  1,  0,  2,  1,  0,  5,  4,  3,  2,  1,  0,
    12, 11, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  1,  0,  7,  6,  5,  4,  3,  2,  1,  0,  5,  4,
     3,  2,  1,  0,  0,  0,  4,  3,  2,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  3,  2,  1,  0,  6,  5,
     4,  3,  2,  1,  0,  0,  4,  3,  2,  1,  0, 18, 17, 16, 15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,
     4,  3,  2,  1,  0, 10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0,  2,  1,  0,  3,  2,  1,  0,  2,  1
    ]
    array_size = len(MP2)
    error_check = int(math.log(array_size, 5)) # The array_size should always be a power of 5.
    current_power = int((modLength + 3)//4)
    current_index = 0
    delta_limit = 1 #620
    o4_solutions = 0
    e2_solutions = 0
    e1_solutions = 0
    if modLength > 3:
        current_index = current_power % array_size
    solutions = 0
    curr_solution = current_power >> 32
    prev_solution = curr_solution
    max_power = 5**(modLength - 1) + int((modLength + 3)//4)
    upper_limit = max_power - delta_limit + 1
    print(modLength)
#   f2 = open(f"{modLength}_MSC2.txt", "w")
    truncPowerOf16 = str(pow(16, current_power, 10**(modLength + 1))).zfill(modLength + 1) # Change power and zfill to modLength
for alternate approach.
    while current_power < upper_limit:
        isMSP = False
        if '1' not in truncPowerOf16[1:]: # Remove [1:] for alternate approach.
            if '2' not in truncPowerOf16[1:]: # Remove [1:] for alternate approach.
                if '4' not in truncPowerOf16[1:]: # Remove [1:] for alternate approach.
                    if '8' not in truncPowerOf16[1:]: # Remove [1:] for alternate approach.
                        match = re.search(pattern, truncPowerOf16[:0:-1]) # Remove 0 for alternate approach.
                        if not match:
                            isMSP = True
        if isMSP:
            solutions += 1
            match = re.search(pattern2, truncPowerOf16[:0:-1]) # Remove 0 for alternate approach.
```

```
            if truncPowerOf16[0] in '02468':
                if match:
                    e1_solutions += 1
                else:
                    e2_solutions += 1
            else:
                o4_solutions += 1
#           print("\t", solutions, current_power, sep = " - ")
#       if ((current_power % 5**(modLength - 2) >= 5**(modLength - 2) - 100) or (current_power % 5**(modLength - 2) < 100)):
#           f2.write(f"{modLength} {current_power} {solutions}\n")
        # The following if statement is needed for modLength <= error_check.  Commented out for speed.
        # Otherwise, indent the first line after it.
#       if modLength > error_check:
        current_index += 1
        current_power += MP2[current_index] + 1
        truncPOf16Temp = str(int(int(truncPowerOf16) * pow(16, MP2[current_index] + 1, 10**(modLength + 1)))).zfill(modLength + 1)
# Change power and zfill to modLength for alternate approach.
        truncPowerOf16 = truncPOf16Temp[(- modLength - 1):] # Remove the -1 for alternate approach.
        current_index = (current_index + MP2[current_index]) % array_size
        curr_solution = current_power >> 32
        if prev_solution != curr_solution:
            # Not an accurate check for percentage left since the upper bound keeps decreasing.
            # It's an issue of speed vs. accuracy.
            percentage_left = round(current_power/upper_limit, 2)
#           percentage_left = round((math.log(current_power, 5))/(modLength - 1), 2)
            print(modLength, solutions, current_power, percentage_left, sep = " - ")
            prev_solution = curr_solution
            f = open(f"{modLength}_MSC.txt", "w")
            f.write(f"{solutions} {current_power}")
            f.close()
#   f2.close()
    print("QED", modLength, solutions, sep = " - ")
    print("   ", modLength + 1, o4_solutions, e2_solutions, e1_solutions, sep = " - ")
#   print("   ", modLength, o4_solutions//4, e2_solutions//2, e1_solutions, sep = " - ") # Use this in alternate approach.

minimalPowersOf2(17)
```

References

[1] Graham Higman (1952), "Ordering by divisibility in abstract algebras", Proceedings of the London Mathematical Society, (3) (1952), **2** (7): 326-336, https://doi.org/10.1112/plms/s3-2.1.326

[2] Jeffrey O. Shallit – "Minimal Primes", *J. Recreational Mathematics* **30** (2) (2001), 113-117, http://www.cs.uwaterloo.ca/~shallit/papers.html

[3] Bassam Abdul-Baki – "Minimal Sets for Powers of 2" (2010), http://www.abdulbaki.org/math/MinimalSets2010.pdf

[4] OEIS Foundation Inc. (2019), The On-Line Encyclopedia of Integer Sequences, http://oeis.org/A071071

[5] Python Programming Language (1990), https://www.python.org/.