Extending A056154

Jonathan Frech
⟨info@jfrech.com⟩

2019-10-28

Abstract

In this paper I will present the results from an exhaustive search of natural numbers $n < 60\,000\,000$ regarding membership in OEIS sequence A056154 written in 1 155 lines of C, both verifying all previously known twelve members and finding a new member,

$$\text{A056154}(13) = 49\,094\,174.$$

Definition of A056154

The On-Line Encyclopedia of Integer Sequences defines sequence A056154 as

> "[n]umbers n such that the number of times each digit occurs in 2^n, represented in base 3, is the same as 2^{n+1}, also represented in base 3. Or in other words, when represented in base 3, the digits in 2^n can be rearranged to form 2^{n+1}."[1]

Currently known sequence terms are

$$\begin{aligned}\text{A056154} = (&5, 27, 40, 92, 138, 929, 1\,086, 352\,664, 4\,976\,816,\\ &9\,914\,261, 23\,434\,996, 30\,490\,425, 49\,094\,174, \dots).\end{aligned}$$

Machine

All calculations were performed on an Intel Core i7-4790K CPU at 4.00 GHz, running 64-bit Linux Mint 19.2 Tina with 8 GB of RAM.
It took this machine 398 375 s ≈ 4.61 d to find the thirteenth sequence member and a total of 600 853 s ≈ 6.95 d to complete the entire search.

[1] `https://oeis.org/A056154`

Methodology

Since the sequence is defined by a property specific to base 3 on natural numbers that far exceed typical native number type ranges, I opted for an internal representation tailored towards ternary inspection.
As an optimization, I chose a representation in base 3^t where $t \in \mathbb{N}^+$, calculating subsequent powers of two by iteratively doubling in a manner optimized for the aforementioned representation, starting at $2^0 = 1$. Digits in this representation will be from now on referred to as "xits".

To achieve further performance benefits, five parameters govern the search's approach:

parameter	value range	description
xit_t	{uint8_t, uint16_t, uint32_t, uint64_t}	data type used for a single xit
xit_carry_t	{uint8_t, uint16_t, uint32_t, uint64_t}	data type used to compute the sum of two xits
rit_count_t	{uint8_t, uint16_t, uint32_t, uint64_t}	data type used to count trits
threepow	{1, 2, ..., 16}	power t determining the representation
pass	{--, --one_pass}	doubling algorithm uses two passes or one pass

Note that larger exponents $t > 16$ are possible. However, since a lookup table is created, the memory footprint quickly becomes unmanageable.

Without restrictions, there are $4^3 \cdot 16 \cdot 2 = 2048$ possible combinations of the above parameters, of which only $284 \cdot 2 = 568$ describe a valid computation.
All valid parameter combinations' temporal performance was measured and averaged over eight runs, each checking the first $n < 10\,000$ natural numbers. The three best and three worst performing parameter combinations were:

time	xit_t	xit_carry_t	rit_count_t	threepow	pass
0.0164 s	uint32_t	uint32_t	uint32_t	11	--one-pass
0.0170 s	uint32_t	uint32_t	uint64_t	11	--one-pass
0.0186 s	uint16_t	uint32_t	uint32_t	10	--one-pass
...	...	...	...	...	...
0.4229 s	uint64_t	uint16_t	uint64_t	1	--
0.4345 s	uint8_t	uint64_t	uint32_t	1	--
0.4372 s	uint8_t	uint64_t	uint64_t	1	--

Based on the above results, I chose the best performing parameter combination to run an exhaustive check on all natural numbers $n < 60\,000\,000$, even though the relative performance ranking may change with regard to n. A possible improvement could be re-running above performance analysis and adapting accordingly during the search.

Verification

Both $2^{49\,094\,174}$ and $2^{49\,094\,175}$ require the same number of the same trits:

trit	#
'0'	10 323 527
'1'	10 325 258
'2'	10 326 191
Σ	30 974 976

When encoding trits with ASCII codes '0' = 48, '1' = 49, '2' = 50 and appending a newline '\n' = 32, the corresponding MD5 hashes are as follows.

MD5(2pow49094174.ternary) = `528b5a45dce6d8fe1e110d83e4845958`
MD5(2pow49094175.ternary) = `c8817751f5db719324ca9917d2f0bd93`

Due to the numbers' size, printing their full ternary representation is not feasible. Therefore, only the first and last fifty trits will be given:

$$\begin{aligned} 2^{49\,094\,174} &= 10100202200011112100212011010201112220000012202002\ldots \\ &\quad \ldots 20222002002222001101011210212222221011021020011011_3 \\ 2^{49\,094\,175} &= 20201112100100001201201022021110002210000102111012\ldots \\ &\quad \ldots 11221011012221002202100121202222212022112110022022_3 \end{aligned}$$

Since $3^{39} \cdot 2 < 2^{64}$, the last thirty-nine trits were verified by computing

$$2^{49\,094\,174} \mod 3^{39} \text{ and } 2^{49\,094\,175} \mod 3^{39}$$

using a native 64-bit unsigned integer implementation.

Shortcomings

As already mentioned, an adaptive parameter search might lead to increased performance. Furthermore, the current memory footprint can get unmanageable if too many intermediate results are stored. Saving to disk also requires the entirety of all computed data to be kept in memory at once, instead of being implemented in a sequential manner.

Source

Source code and makefile needed to compile the binary used to perform the search presented in this paper are hosted on `papers.jfrech.com` and licensed under the MIT license:

- `https://papers.jfrech.com/2019-10-28_jonathan-frech_` ↷ `extending-a056154/source.tar`
- `https://papers.jfrech.com/2019-10-28_jonathan-frech_` ↷ `extending-a056154/ternary-data.tar`