Problem 14

The following iterative sequence is defined for the set of positive integers:

$$n \to n/2 \text{ (n is even)}$$

$$n \to 3n+1 \text{ (n is odd)}$$

Using the rule above and starting with 13, we generate the following sequence:

$$13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

Which starting number, under one million, produces the longest chain?

NOTE: Once the chain starts the terms are allowed to go above one million.

The obvious solution is to simply brute-force every value by calculating its chain. The following program computes Collatz(n), the length of the chain starting with n, for all n below 10**6. It runs in 58.75 seconds in Python. [1]

```
FUNCTION countChain(n)
   IF n = 1 THEN
      RETURN 1
   IF n MOD 2 = 0 THEN
      RETURN 1 + countChain(n / 2)
   ELSE
      RETURN 1 + countChain(3 * n + 1)
   ENDIF
ENDFUNCTION

longest_chain ← 0
answer ← -1

FOR number ← 1 TO 10^6 - 1
   IF countChain(number) > longest_chain THEN
      longest_chain ← countChain(number)
      answer ← number
   ENDIF
ENDFOR
OUTPUT answer
```

However, there are obvious inefficiencies in this approach.

1. Many values are being recalculated. The second value in Collatz(26) is 13, after which we recalculate the entire chain seen above. It is much faster to store Collatz(13) = 10, and compute Collatz(26) = 1 + Collatz(13) = 11. This crucial optimization reduces the runtime from 58.75 seconds to 2.33 seconds.
2. If n is even, then $n \to n/2 \Rightarrow$ Collatz(n) = Collatz(n/2) + 1. Therefore Collatz(2k) > Collatz(k) for all k, and we do not need to compute the chain for any $k \leq LIMIT/2$. In this case, we do not need to compute the chain for any k below 500000.
3. If n is odd, then 3n + 1 is even, and $n \to 3n+1 \to \frac{(3n+1)}{2}$. We can save a step by giving Collatz(n) = Collatz($\frac{(3n+1)}{2}$) + 2.

[1] All programs were run on a 1.3 GHz Intel Core i5 processor.

The following program implements all of the above optimizations, and runs in roughly 1.5 seconds in Python:

```
FUNCTION countChain(n)
   IF EXISTS(values[n]) THEN
      RETURN values[n]

   IF n MOD 2 = 0 THEN
      values[n] = 1 + countChain(n / 2)
   ELSE
      values[n] = 2 + countChain((3 * n + 1) / 2)
   ENDIF

   RETURN values[n]
ENDFUNCTION

longest_chain ← 0
answer ← -1
values ← {1: 1}

FOR number ← 500000 TO 10^6 - 1
   IF countChain(number) > longest_chain THEN
      longest_chain ← countChain(number)
      answer ← number
   ENDIF
ENDFOR
OUTPUT answer
```

Another tempting "optimization" is to only consider odd numbers as possible solutions. Observing which numbers are new maxima (i.e. they generate a longer chain than any lower value) suggests this is valid: after 2, 6, and 54, there are no further even maxima below 10**6. However, this is not a guarantor of future behavior, and it turns out there *are* more even maxima: the next two are 31466382 and 537099606. If the upper bound were 32000000, skipping even numbers would miss the solution.