Collatz Conjecture for $2^{100000} - 1$ is True - Algorithms for Verifying Extremely Large Numbers

Wei Ren[a,b]

[a]School of Computer Science,
China University of Geosciences, Wuhan, China
[b]Hubei Key Laboratory of Intelligent Geo-Information Processing
(China University of Geosciences (Wuhan)), Wuhan, Hubei, China

Abstract

Collatz conjecture (or 3x+1 problem) is out for about 80 years. The verification of Collatz conjecture has reached to the number about 60bits until now. In this paper, we propose new algorithms that can verify whether the number that is about 100000bits (30000 digits) can return 1 after 3*x+1 and x/2 computations. This is the largest number that has been verified currently. The proposed algorithm changes numerical computation to bit computation, so that extremely large numbers (without upper bound) becomes possible to be verified. We discovered that $2^{100000} - 1$ can return to 1 after 481603 times of 3*x+1 computation, and 863323 times of x/2 computation.

Keywords:
Collatz Conjecture; 3x+1 Problem; Computational Algebra; Algorithmic Number Theory;

1. Introduction

The Collatz conjecture is a mathematical conjecture that is first proposed by Lothar Collatz in 1937. It is also known as the 3x+1 conjecture, the Ulam conjecture, the Kakutani's problem, the Thwaites conjecture, or the Syracuse problem [2].

Simply speaking, the conjecture can be stated as follows. Take any positive integer number x. If x is even, divide it by 2 to get $x/2$. If x is odd,

Email address: `weirencs@cug.edu.cn` (Wei Ren)

multiply it by 3 and add 1 to get $3 * x + 1$. Repeat the process again and again. The Collatz conjecture is that no matter what the number (i.e., x) is taken, the process will always eventually reach 1.

The contributions of the paper are listed as follows:

1. A new computing algorithm is proposed that can verify Collatz conjecture up to a new upper bound. The proposed algorithm changes numerical computation to logical computation, and compute $3 * x + 1$ bit by bit in binary. This upper bound is significantly larger than any existing known algorithms and experimental witness. The current known upper bound for starting value that has been checked (or can be checked) is up to about 60bits [1, 2], but the proposed algorithm in this paper can check starting numbers up to 100000bits.
2. A new computing algorithm is proposed that can verify Collatz conjecture up to theoretically unlimited upper bound, only depending on the timing cost. The rationale in the algorithm is using outer memory (such as a data file in a hard disk) instead of inner memory (such as an array or allocated memory) for data processing.

The rest of the paper is organized as follows. Section 2 presents relevant background. Section 3 details our proposed algorithms and analysis. Section 4 provides some experimental results. Finally, Section 5 concludes the paper.

2. Preliminary

Definition Collatz Transformation

$$CT(x) = \begin{cases} TPO(x) = 3 * x + 1 \quad (TPO) \quad (x \in [1]_2), \\ H(x) = x/2 \quad (H) \quad (x \in [0]_2), \end{cases} \tag{1}$$

where $[1]_2 = \{a | a \equiv 1 \mod 2, a \in \mathbb{N}\}$, $[0]_2 = \{a | a \equiv 0 \mod 2, a \in \mathbb{N}\}$. $\mathbb{N} = \{a | a \in \mathbb{Z}, a \geq 1\}$. $TPO(x)$ can be simply denoted as TPO, and $H(x)$ can be simply denoted as H.

Definition The Collatz Conjecture. $\forall x \in \mathbb{N}$. After finite times of Collatz Transformation $x \Leftarrow CT(x)$, x will become 1. (Here "$\Leftarrow$" is assignment symbol and "$x \Leftarrow y$" means to assign value y to x.)

Example The dynamics after each Collatz Transformation from a starting value (at head) to 1 (at rear) are as follows:

(1) $1 \rightarrow 4 \rightarrow 2 \rightarrow 1$;

(2) $7 \rightarrow 22 \rightarrow 11 \rightarrow 34 \rightarrow 17 \rightarrow 52 \rightarrow 26 \rightarrow 13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$.

3. Proposed Algorithms

Two notations will be used in the paper as follows:

1. (x) returns the least significant bit (LSB) of binary number x.
2. $\langle x \rangle$ returns the most significant bit (MSB) of binary number x.

Simply speaking, if $x \leq 3$, (x) returns rightmost bit of binary representation of x, and $\langle x \rangle$ returns leftmost bit of binary representation of x.

E.g., $(10) = 0$, $\langle 10 \rangle = 1$, where 10 is a binary number;
Besides, $(1 + 0) = 1, (1 + 1) = 0, \langle 1 + 1 \rangle = 1, \langle 1 + 0 \rangle = 0$.

Fig.3 depicts the design rationale in the proposed algorithm. Suppose the length of binary representation on x is n. $a[0]$ is MSB (leftmost bit) of x, and $a[n-1]$ is LSB (rightmost bit) of x. $2 * x$ is one bit of left shift. The LSB of $2 * x$ is thus 0. $3 * x + 1$ can be looked as $x + 2x + 1$. Suppose the resulting summation is $b[0]b[1]...b[n-1]$. The carrier bit is $c[0]c[1]...c[n-1]$. As x is odd (due to $3 * x + 1$ is to be computed), $a[n-1] = 1$. The rightmost bit of result (i.e., $3 * x + 1$) is $(a[n-1] + 0 + 1)$, which is $(1 + 0 + 1) = 0$. The carrier bit at this location is $c[n-1] = \langle a[n-1] + 0 + 1 \rangle = \langle 1 + 0 + 1 \rangle = 1$. Next, $b[n-2] = (a[n-2] + a[n-1] + c[n-1])$, where $a[n-1] = 1$ and $c[n-1] = 1$. The following computation of the other bits is $b[n-k] = (a[n-k+1] + a[n-k] + c[n-k+1])$, $c[n-k] = \langle a[n-k+1] + a[n-k] + c[n-k+1] \rangle$. See Eq.2 and Eq.3 for further details.

Major computational logics in the proposed algorithm are listed in Eq.2 and Eq.3. Note that, before the computation of $3 * x + 1$, x is firstly represented as binary string like $a[0] \| a[1] \| ... \| a[n-2] \| a[n-1]$ from MSB to LSB. The computation is from LSB to MSB.

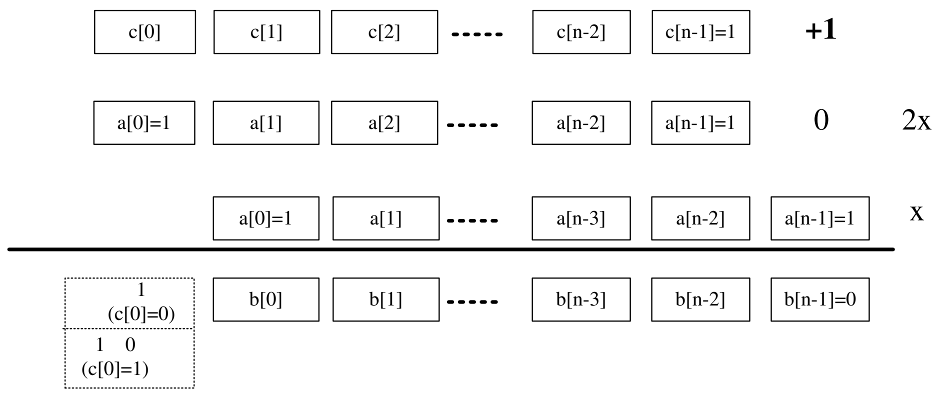

Figure 1: Fast algorithm for $3 * x + 1$ (TPO) on an extremely large number x via bits computation instead of numerical computation.

$$\left\{\begin{aligned} b[n-1] = (a[n-1]+0+1) = (1+0+1) = 0 \\ c[n-1] = \langle a[n-1]+0+1\rangle = \langle 1+0+1\rangle = 1, \\ b[n-2] = (a[n-1]+a[n-2]+c[n-1]) = (a[n-1]+a[n-2]+1) \\ c[n-2] = \langle a[n-1]+a[n-2]+c[n-1]\rangle = \langle a[n-1]+a[n-2]+1\rangle, \\ \ldots \\ b[n-k] = (a[n-k+1]+a[n-k]+c[n-k+1]) \\ c[n-k] = \langle a[n-k+1]+a[n-k]+c[n-k+1]\rangle, \\ \ldots \\ b[1] = (a[1]+a[2]+c[2]) \\ c[1] = \langle a[1]+a[2]+c[2]\rangle, \\ b[0] = (a[0]+a[1]+c[1]) = (1+a[1]+c[1]) \\ c[0] = \langle a[0]+a[1]+c[1]\rangle = \langle 1+a[1]+c[1]\rangle. \end{aligned}\right. \tag{2}$$

$Head$ is the font one bit or two bits of $3 * x + 1$, depending on $c[0] = 0$ or not. $\underline{1}$ is one bit; $\underline{10}$ are two bits. The final result of $3 * x + 1$ is represented as binary string like $Head\|b[0]\|b[1]\|...\|b[n-2]\|b[n-1]$, where $Head$ is one bit "1" or two bits "10". $\|$ is concatenation.

$$Head = \begin{cases} c[0]+a[0] = c[0]+1 = 0+1 = \underline{1}, & c[0] = 0, \\ c[0]+a[0] = c[0]+1 = 1+1 = \underline{10} & c[0] = 1. \end{cases} \tag{3}$$

According to Eq.2 and Eq.3, the proposed algorithm (Algorithm 1) is given as follows:

```
Data: $x$
Result: $TPO(x)$. That is, $result = TPO(x) = 3 * x + 1$.
$b[n-1] \Leftarrow' 0';$
$c[n-1] \Leftarrow 1;$
for $(i = n-2; i >= 0; i--)$ do
    $sum \Leftarrow a[i+1] + a[i] + c[i+1];$
    if $sum == 2 || sum == 3$ then
        $c[i] \Leftarrow 1;$
    end
    if $sum == 0 || sum == 1$ then
        $c[i] \Leftarrow 0;$
    end
    if $sum == 0 || sum == 2$ then
        $b[i] \Leftarrow' 0';$
    end
    if $sum == 1 || sum == 3$ then
        $b[i] \Leftarrow' 1';$
    end
end
if $c[0] == 1$ then
    $result \Leftarrow'' 10'' \| result;$
end
if $c[0] == 0$ then
    $result \Leftarrow' 1' \| result;$
end
return $result$;
```

Algorithm 1: Input an extremely large number x that is represented in binary. Output $TPO(x)$ that is $3 * x + 1$.

Algorithm 1 changes numerical computation of $3 * x + 1$ into simple bit computation. That is, only addition of three bits is conducted (in $sum \Leftarrow a[i+1] + a[i] + c[i+1]$), avoiding to compute $3 * x$ directly in which x is upper-bounded by memory limitation in computer. Thus, it becomes possible to compute $3 * x + 1$ of an extremely large number by our algorithm.

The numerical computation $sum \Leftarrow a[i+1]+a[i]+c[i+1]$ can be changed to logical computation as follows (Algorithm 2):

Thus, Algorithm 2 computes a starting number x with binary length n in $O(n)$ time. Besides, in each time of loop execution (i.e., For), only one decision instruction (i.e., If) is executed. The result (i.e., $3*x+1$) can be computed in one pass traverse of x from LSB (i.e., $a[n-1]$) to MSB (i.e., $a[0]$).

Above algorithms are used for outputting and analyzing the dynamics of Collatz transformations on extremely large inputting numbers, e.g., outputting whole procedure from starting number that is an extremely large number to final number that is 1. The invoking of $TPO(x)$ function is given in Algorithm 3 as follows:

Algorithm 3 outputs dynamics called *code*. The *count_up* is the occurred times of "up", and *count_down* is the occurred times of "down" in dynamics (or *code*). Next, we explain what are "up" and "down".

The procedure (or dynamics) from starting number to 1 consist of two Collatz transformations or computations, which is $3*x+1$ and $x/2$. The dynamics are recorded by Algorithm 3 in terms of a serial of Collatz transformations. $3*x+1$ is denoted as $TPO(\cdot)$ and $x/2$ is denoted as $H(\cdot)$. That is, $TPO(x) = 3*x+1, H(x) = x/2$. We are aware following fact.

Proposition 3.1. *If TPO occurs, H occurs intermediately after it. That is, "TPO, H" always together occurs in any dynamics.*

Proof If $x \in [1]_2$, $TPO(x) = 3*x+1 \in [0]_2$. Thus, the next Collatz transformation immediately after "TPO" must be "H". Therefore, H always occurs after a TPO transformation. That is, "TPO" and H always *together* occurs. □

Therefore, $(3*x+1)/2$ can be written together as $H(TPO(\cdot)$, and denoted as "$-$" in outputting code in our computer program. For better contrast and outputting vision, we denote $H(x)$ as "0" in outputting code in our program.

For example, the outputting code (or dynamics) for $x = 5$ is "-000", which means $H(TPO(\cdot))$, $H(\cdot)$, $H(\cdot)$, $H(\cdot)$.

In other words, $5 \to 16 \to 8 \to 4 \to 2 \to 1$.

As $3*x+1 > x$, we call $TPO(x)$ as "up" computation. As $x/2 < x$, we call $H(x)$ as "down" computation. The times of "up" equals the count of "$-$". For example, the times of "up" in *code* for 5 is 1, as there exists one "$-$" in *code*. Note that, as in $(3 * x + 1)/2$ "down" also occurs, its times is also the count of "$-$". Thus, the total times of "down" equals the summation of two parts - the count of $-$ and the count of "0" in outputting *code*. For example, the times of "down" in *code* for 5 is 4, because there are 3 "0" in *code* and 1 "$-$" in *code*.

Algorithm 3 outputs *code* that represents dynamics, which consists of "$-$" and "0", to represent occurred Collatz transformations in the procedure from starting number x to 1. The *count_up* is the total times of "up" computation, or $3 * x + 1$ computation, which equals the count of "$-$" in outputting *code*. The *count_down* is the total times of "down" computation, or $x/2$, which equals the summation of two parts - the count of "0" and the count of "$-$". Thus, the count of "0" can be looked as $count\_down - count\_up$, or gap between "down" and "up". The *ratio* can be looked as the gap over times of "up" to measure the major characteristics of dynamics, which indeed is the count of "0" over the count of "$-$".

Besides, to support extremely large numbers that is even in binary, we need to break the limitation of inner memory in computer (e.g., the array size, the size of data type such as "long long integer"). Therefore, in the implementation of Algorithm 3, to break the limitation of inner memory, we use a data file in hard disk to process those extremely large numbers (see Algorithm 4 in Appendix). That is, the reading and writing operation are changed to "from a data file in hard disk", instead of "from an data array or an allocated memory in inner memory". Thus, it can support any length of manipulated number without any upper bound (i.e., starting number x and occurred intermediate numbers during the procedure can be unlimited large).

4. Experiment Results

Example Some examples for dynamics from starting number x to final number 1 are listed as follows ($-$ represents $H(TPO(x))$, or $(3 * x + 1)/2$; 0 represents $H(x)$, or $x/2$):

(1) $x = 7$, $(x)_2 = 111$. $(x)_2$ means x is represented in binary.

Dynamics (intermediate numbers after Collatz transformations):

$1011 \to 10001 \to 11010 \to 1101 \to 10100 \to 1010 \to 101 \to 1000 \to 100 \to 10 \to 1$.

Thus, code is "$---0-00-000$". $U = 5$, $D-U = 6$, $ratio = (D-U)/U = 1.2000000$.

(2) $x = 63$, $(x)_2 = 111111$.

Code is "$- - - - - - 000 - -0 - - - 0 - - - -0 - 00 - - - 0 - -0 -$ $- - - - -00 - - - -000 - 0 - 0 - 000 - 00 - - - 0000 - 000$".

$U = 39$ $D - U = 29$ $ratio = (D - U)/U = 0.7435898$.

Example W.l.o.g., we select numbers with binary form liking 1^{MAXLEN} as testing examples. E.g., $1^{100} = \underbrace{111...11}_{100}$, $1^{1000} = \underbrace{111...11}_{1000}$, and so on.

The dynamics of the largest number x that has been computed until now is as follows:

$(x)_2 = 1^{100000} = \underbrace{111...1}_{100000}$, where $(x)_2$ means x in binary.

In its dynamics, the total times of "up" computations (i.e, $3 * x + 1$) or the count of "$-$" in outputting code is 481603; the total times of "down" computations (i.e., $x/2$) is the count of "0" in outputting code, together with the count of "$-$" in outputting code. That is, $381720 + 481603 = 863323$. The ratio is 0.7926030 (namely, $ratio = (D - U)/U = 381720/481603 = 0.7926030$. In other words, this number endures 481603 times of TPO Collatz transformation, and 863323 times of H Collatz transformations to become 1.

The details of data is provided as supplement (upon request by email or on my personal web site), as it is too long to present in the paper.

The dynamics is from starting number to 1. The count of computations is listed in Tab. 1. (Besides, in our another paper, we proved the bound of *ratio*.)

5. Conclusion

In this paper, we proposed several algorithms that can be used for verifying Collatz conjecture to extremely large numbers. The advantages of the proposed algorithms are as follows: (1) The computation is only symbol logical computation (i.e., $\wedge$, $\vee$), which is very fast comparing with numerical computation. (2) The computation of $3 * x + 1$ for extremely large number x becomes possible, as x is represented in binary and computed bit by bit. (3) The computation has no memory limitation. Theoretically, $x = 2^{+\infty}$, as intermediate and final computation results are written to a file in hard disk instead of to an array or allocated memory in inner memory. The experimental results show that current the largest binary number with the form

$1^{100000} = \underbrace{111...11}_{100000}$ can return to 1, after 481603 times of TPO Collatz transformation, and 863323 times of H Collatz transformations.

Acknowledgement

The research was financially supported by the National Natural Science Foundation of China (61170217).

References

[1] Silva, Tomas Oliveira e Silva. Computational verification of the 3x+1 conjecture. *http://sweet.ua.pt/tos/3x+1.html*, Retrieved 28 Jan. 2016.

[2] Wikipedia. Collatz conjecture. *https://en.wikipedia.org/wiki/Collatz_ conjecture*, Retrieved 28 Jan. 2016.

```
Data: $x$
Result: $TPO(x)$. That is, $result = TPO(x) = 3 * x + 1$.
$b[n-1] \Leftarrow' 0';$
for $(i = n-2; i >= 0; i--)$ do
    if $(a[i+1] == 1)\&\&(a[i] == 1)\&\&(c[i+1] == 1)$ then
        $c[i] \Leftarrow 1;$
        $b[i] \Leftarrow' 1';$
    end
    if $(a[i+1] == 1)\&\&(a[i] == 0)\&\&(c[i+1] = 1)$ then
        $c[i] \Leftarrow 1;$
        $b[i] \Leftarrow' 0';$
    end
    if $(a[i+1] == 0)\&\&(a[i] == 1)\&\&(c[i+1] = 1)$ then
        $c[i] \Leftarrow 1;$
        $b[i] \Leftarrow' 0';$
    end
    if $(a[i+1] == 1)\&\&(a[i] == 1)\&\&(c[i+1] == 0)$ then
        $c[i] \Leftarrow 1;$
        $b[i] \Leftarrow' 0';$
    end
    if $(a[i+1] == 1)\&\&(a[i] == 0)\&\&(c[i+1] == 0)$ then
        $c[i] \Leftarrow 0;$
        $b[i] \Leftarrow' 1';$
    end
    if $(a[i+1] == 0)\&\&(a[i] == 1)\&\&(c[i+1] == 0)$ then
        $c[i] \Leftarrow 0;$
        $b[i] \Leftarrow' 1';$
    end
    if $(a[i+1] == 0)\&\&(a[i] == 0)\&\&(c[i+1] == 1)$ then
        $c[i] \Leftarrow 0;$
        $b[i] \Leftarrow' 1';$
    end
    if $(a[i+1] == 0)\&\&(a[i] == 0)\&\&(c[i+1] == 0)$ then
        $c[i] \Leftarrow 0;$
        $b[i] \Leftarrow' 0';$
    end
end
if $c[0] == 1$ then
    $result \Leftarrow'' 10'' \| result;$
end
if $c[0] == 0$ then
    $result \Leftarrow' 1' \| result;$
end
return $result;$
```

Algorithm 2: Input an extremely large number x that is represented in binary. Output $TPO(x)$ that is $3 * x + 1$.

```
Data: $x$
Result: $code, count\_up, count\_down - count\_up, ratio$
$IsEven(input)$;
$TXPO(input, result)$;
while $current \neq 1$ do
    if $IsEven(current) \neq 1$ then
        $result \Leftarrow TXPO(current)$;
        % It is intermediate one “down” (x/2) after “up” (3*x+1) ;
        $flag\_first\_down \Leftarrow 1$;
        $count\_up + +$;
        $continue$;
    end
    else
        if $flag\_first\_down == 1$ then
            $code \Leftarrow code \| -$;
            $count\_down + +$ ;
            $flag\_first\_down \Leftarrow 0$;
        end
        else
            $code \Leftarrow code \| 0$;
            $count\_down + +$ ;
        end
        $continue$;
    end
end
return $code, count\_up, count\_down - count\_up, ratio =$
  $(count\_down - count\_up)/count\_up$;
```

Algorithm 3: Input an extremely large number x. Output $code$, $count\_up$, $count\_down - count\_up$, $ratio = (count\_down - count\_up)/count\_up$.

Table 1: Times of "up" computations is denoted as U. Times of "down" computations is denoted as D. $ratio = \frac{D-U}{U}$. x is in binary with the form 1^{MAXLEN}, and its length is $MAXLEN$. The dynamics is recorded from starting number to 1.

$MAXLEN$	x in binary	digits of x	$(U, D-U)$	$ratio$
100	$1^{100} = \underbrace{111...1}_{100}$	30	(528, 409)	0.7746212
500	$1^{500} = \underbrace{111...1}_{500}$	150	(2417, 1914)	0.7918908
1000	$1^{1000} = \underbrace{111...1}_{1000}$	300	(4316, 3525)	0.8167285
5000	$1^{5000} = \underbrace{111...1}_{5000}$	1500	(24131, 19116)	0.7921761
10000	$1^{10000} = \underbrace{111...1}_{10000}$	3000	(48126, 38152)	0.7927524
50000	$1^{50000} = \underbrace{111...1}_{50000}$	15000	(239020, 189818)	0.7941511
100000	$1^{100000} = \underbrace{111...1}_{100000}$	30000	(481603, 381720)	0.7926030

Appendix

Algorithm 4.

Example Some examples for dynamics from starting number x to final number 1 are listed as follows ($-$ represents $H(TPO(x))$, or $(3 * x + 1)/2$; 0 represents $H(x)$, or $x/2$):

(1) $x = 2^{100} - 1$. $(x)_2 = 1^{100} = \underbrace{111...11}_{100}$.

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−0000−00−−−00−000−−0−
−000−000−−0000−0−−00−−−00−0−−−00−0−−−0−−−0−00−−−
0−0−−−0000−−00−0−00−−−0−0000−−−00−−−−−0−00−0000−
000−−000−000−−−0−00−00−0−−00−−−−−−0−−−−−00−−−0−
0−0000−0−−000−0−−−−0−00000−−−000−0−0000−0−−−−−−00−
−00−00−−−0−0−00000−0−0000−0−−−−00−−−−−00−−0−−0−
000000−−0−−0000−00−00−−−00−−0−−0−−000−0−−−−0−−−−−
−0−−−−00−0000−−−0−−−0−00−−0−−−000−00000−00−−−−−
0−−0−−0−0−−−0−−0−−−−−−000−00−−−00−0−0−0−−−0−00−
0000−00−−−0−0−0−−00−00−00−0−−0−00−0000−−−−−−0−−−
00−−0000−−−−−0−0−0−0−−−−00−0−−0−0−−−−0−−000−−−
−−−0−00−−0−−−00−00−0−00−−−0−−00−−0−000−000−0−−−−−
00−−−−−0−−−0−−0−0−0−000−−0−−000000−−000−−0−0−−0−
0−0−00−−−−0−−00−0−0−−0−−−000000−−−0000−−0−00−−−
−0−0−0−−000−0000−0−−0−−−−−−0−−000−0−−00−00−0−00−
0−−−0−0−00−00−000−−−−−00−−−−−00−−−0−0−00000000−−−
−0000−0−−−−−−0−−0−0−0−0−−−−0−0−0−00−0−0−−0−−00−
00−0000−0−−0−−0−00−0−−000000−0−00−00−0000−−−0000−000.

(2) $x = 2^{500} - 1$. $(x)_2 = 1^{500} = \underbrace{111...11}_{500}$.

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

```
Data: The file with the file name "$TESTNUMBER$"
Result: The file with the file name "$CODE$"
$GenTestNumber(''TESTNUMBER'');$
$FileCopy(``TESTNUMBER'','' input\_start'');$
$FileCopy(``TESTNUMBER'','' current\_x'');$
while $(GetFileLength(''current\_x'')! = 1)$ do
    if $IsEven(''current\_x'') == 0$ then
        $TXPO(''current\_x'','' temp'');$
        $FileCopy(''temp'','' current\_x'');$
        $Firstdown \Leftarrow 1;$
        $count\_up \Leftarrow count\_up + 1;$
        $continue;$
    end
    else
        if $Firstdown == 1$ then
            $CutFileRear(''current\_x'');$
            $AppendToFile(''CODE'','-');$
            $count\_down \Leftarrow count\_down + 1;$
            $Firstdown \Leftarrow 0;$
            $FileAppend(''current\_x'','' DYNAMICS'');$
        end
        else
            $CutFileRear(''current\_x'');$
            $AppendToFile(''CODE'','0');$
            $count\_down \Leftarrow count\_down + 1;$
            $FileAppend(''current\_x'','' DYNAMICS'');$
        end
        $continue;$
    end
end
$ratio \Leftarrow (count\_down - count\_up)/count\_up;$
return$(count\_up, count\_down, ratio);$
```

Algorithm 4: Main algorithm for outputting dynamics. Output $CODE(x)$ to a file named "$CODE$", where x is the starting number that is the content of the file named "$TESTNUMBER$".

−−
−−
−−
−−
−−−−−−−−−−−−−−−−−−−−0000−0−00−−00000−−−−0−−−00−
−−00−−−−00−0−00−−000−−0−−000−−000−−0−−−0−−−0−00000−
0−−−00−−000−0−00−−00−−000−0−−−0−−−00000−−−0−−00−
00−−−00−0000−0−−00−−0−000−00−0−−0−0−00−00−−−00−−00−
000−00000−−−−−00−00−0−0−−−−−00−00−00−−−00−−0−−0−−−
−−−−−00−−00−000−0−0000−−00−0−−−−0−000−−−000−−000−
−00−−−00−−−0−0−0−000−0−0−−0−0−0000−−00−0−−0−−−000−
00−−−−−0000000000000−00−0−−0−−00−0−0000−−0−000−00000−
−0−−0−0−−0−0−−0−−0−0000000−−0−−−00−0−000000−−0000−
00−0−0−0−0−−−00−−−0−−00−−0−0−00−−−0−−0−0−−0000−0−
0−0−000−0−00−0−−0−0−0−0−00−−−0−−0−−−000000−−−−−−0−
0−0−−0−−0−00−−00−−−00−−−−000−−00−−00−−0−0−000−00−
0−−−0−0−00000000−−00−00−0−−−00−0−−−−0−−0−0000−−0−
−−0000−−−−−−−−000−−00−0000−0−−−0−00−0−−−−−−−00−
−000−0−0−−0−−−0−−−0−−−0−−−00−0−−−00−0−−−0−0−00−
−0−−0−−−00−00−−−0−00−0−−0−0000−00−000−−−−0−−000000−
0−−−0−000−000−−00000000−−−−−−−0−−−00−−0−00−0−−0−
−0−00000−0−−−−−00000−−−−−−−−−0−0−−−−−00−0−−−−00−
0−00−−00−−−−−−00−−0−−−−−−−00000−−−−−00−00000−0−
0−0−0−−−0−0−−−−0−−−0−0−0−0−0−−−−−00−−0000−−−0−
00−0−0−00−00−−000−00−−0000−0−0−000−0−−000−−000−0000−
0−−00000−00−−−−−00−0000000−00−000−0−−00−00−−0−−−00−
0−00−000000−0−−00−−−−0−0−−0−00−−−0−0−0−000−−0−−00−
0−0−−0−−0−−−000−−000−−−0000−000−0−00−−00000−00−−−0−
000−−000−0−00−0000−0−0−−0−0−−00−000−−−−0−−−000−−0−
0−−−−0−0−−0−−−−−0000−−−−−−−−0−0−0−−−000−−0−−0−
000000−00−0−−0−−−00−00−0−0−−−00−0−0−0−00000−−0−0−0−
00−0−0−0−0−00000−000−00−00−−−0000−000−0−−0−−−−−−0−
0−−0−−−−−−0−−−0−000−−000−−0−−0−−0000−−−−−0000−−−
000−−00−0−−0000−00−0−0−00−0−0000−00−−−0−0−−−0−−00−
0−−0−−0−00−0−0−00−0−00000−−00−−0−−−−00−−−0−−−0000−
−00−0−−0−0−00−000−−−−000−00−0−−−00−−−000−00−0−0−
0000−−0−0000−−−−00−−−−−0−00−00−0−00−−−0−000−−00−0−
00−0000−000000−000−−00−000−00−0−−0000000−−0−−0000−00−−−

00−−00−−−0−0−−0−00−0−−0−−−0−0−−−−0000−0−−−00−−−
−0−−000−−−0−−00−0−−−00−00−−0−−0−−000−−0−0−0000−−0−
−−000−−−0−0−−−−0−0−0−−0000−−−−−000−−00−−−−0000−
−0000−000−−−−0−000−−−−000−−0000−−0−−−−0−−0−0−0−00−
−00−−−0−00−−−−−0−−000−00000−00−000−0−−00−00−00−−−
−00−−−00−−−−−0−−−00−−00−0−0−−−0−000−00−−0−−−−−
−0−00−−00−−0−−0−0−0−−−0000−0−0−−00−−−00−000−−−00−
0−−−0−00−00−00−−−000−0−−−0000−0−−00−00−0−00−0−0−00−
0−−0−−−000−00−00−0−−0−0−−−0−00−0−0−−−−0−00−0−00−0−
−00−−−−00−0000−00−0−−−0−−000−00−00000−0−0−0000−0−−−
−−−−00−0−−−0−−00−−00000−0−000−−0−0−000−0−00−0−00−
000−0−−00−−−−−000−−0000−−0−−00−−00−−−00−00−−00−0−
−0000000−−−00−−−−−−−0−−−−−−−00−00−−00−0−−0−−−0−
00−−0−−00−−−000−−−−−00000−−−−0000−0−−0−−−000−0−00−
−−000−−−−−0−−00−−−00−−−00−0−−00−−−−−−000−00−−00−
0−00000000−−0−−0−0−0−0−−00−−000000000−−−000−−0−0−0−
−0−−0−00000−−−−−0−0−−000000000−−−00−−−−00−0−000−0−
0−−−−−000000000−000000−00−−−−00−00000−−0−00−−−−−00−
0−−−−00000−−0−000−0−000−−−0−−−00−−−0−0−0−−−00000000−
−−0−000−00−00−00−00−0000−−0−0−0−−−−00−−0−−00−−000−
00−00−00−0−0−−−0−000−−−−−0−0−−−−−0−0−−−0−−−−00−
−−−0−0−00−−−−0−−000−0−−−−−−−−0−−−0−00−00−0−−0−0−
−−−0−0−0−0000−−0−−0−0−−−−0−−−−−00−−−−0−0−−000000−
00−0−0−−0000−−−−−−00−−0−−0−00−0−0−−−0−−0−0−0−00−
−0−−−−−−00−0−000−−−0−−0−−−00−000−0−0−−0−0−0−−0−
−−0−−0−−00−−−0−−−00000−00−0−0−000−−−−0−0000000−−−
00−−−−000−−−−0−−000−00−−−−0−−−0−−−0−−−−0−−−−−0−
−−−0−000−0−000−−−−00000−−−−0−000−−−−−0−0−−−000−00−
00−0−00−00−0000−0−−−00−00−00−−−−−00−−−−−−−0−−−−−
0−0−00−−0−0−−−−−000−−−−0−−0−00000−−−−−−0000−−−0−
−−0−000−0000−−000−−−0−00−−0−−00−0−−000−−00−−−−0−−−
−−0−−0−−−−−00−00−−0000000−00−0−00−−00−−−−000−0−0−
−00−0−−−0−0−−−−−−00−−−000−−0−−0000000−00−0000−−0−
−0−000−−0−0−000−−00−−−−−−000−000−−−000−0000−−−0000−
0−−−0−000−−00−−000−−−0−0−0000−00−000−−0−00−−0−−00−
0000−−00−000−−−−0−00−−−−0−−−−−−0000−0−−−−0−0−−0−
00−0−0−000−0−−00−00−0−0−−000−0−0−0−−000−−000−00000−−−
0−−0−0−00−−−−−−0−−0−00−00−00−0−000000−−−−−000−−−0−

−00−00−00−0−−0−−0−−−0−−00−−00000−−0−00000−−−00−−−−−
−0−0−−−−−−−0−−00−00−0000−−0−−0−−00−−0−0000−0000000−
−0−−00−−0−−−0−−000−0−−00−−−−0000−0−−−−00−−−000−
0000000−−−00−−−−000−00−000000−−−−−0−−000−0000−−−00−
−−−−0−−0−00−0−0−0−−0−−−0−−−−−0−−0−0−−00−−−000−0−
−−−000−000−−000−−00−−−0−00−−0−00−00000−00−−−0000−000.

More examples will be given in supplement materials or upon request.