Efficient Algorithm for Multiplication of Numbers in Zeckendorf Representation

Tomasz Idziaszek
Independent Researcher, Poland
http://algonotes.com
tomasz@algonotes.com

Abstract

In the Zeckendorf representation an integer is expressed as a sum of Fibonacci numbers in which no two are consecutive. We show $O(n \log n)$ algorithm for multiplication of two n-digit numbers in Zeckendorf representation.

For this purpose we investigate a relationship between the numeral system using Zeckendorf representations and the golden ratio numeral system. We also show $O(n)$ algorithms for converting numbers between these systems.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Fibonacci numbers, Zeckendorf representation, multiplication algorithm, Fast Fourier Transform, golden ratio numeral system, Lucas numbers

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.16

1 Introduction

Zeckendorf [12] showed that each non-negative integer has a unique representation as a sum of Fibonacci numbers in which no two consecutive Fibonacci numbers occur. This observation leads to a numeral system. One of its applications is in self-delimiting codes [2], but we can research this system for its own sake as a mathematical curiosity.

A natural question for a numeral system is how can we perform arithmetic operations on numbers in such a system, and how fast can we do it. It was shown that addition and subtraction of n-digit numbers in the Zeckendorf system can be performed in $O(n)$ time [1], so as fast as in the binary system.

But multiplication seems to be much harder. It is straightforward to directly utilize addition and construct a grade-school-like $O(n^2)$ multiplication algorithm for the Zeckendorf system, but so far no one presented a faster method. However, for the binary system we can devise $O(n \log n)$ algorithms [9], by observing that multiplication can be reduced to a convolution (which can be calculated using the Fast Fourier Transform) followed by a normalization phase (carry propagation).

The purpose of this paper is to present $O(n \log n)$ algorithm for multiplication of two n-digit numbers in the Zeckendorf system. The main idea is to reduce it to a convolution followed by a certain normalization phase. The normalization phase can be reduced to $O(\log n)$ additions.

For convolution to work, we use the fact that the Zeckendorf system is closely related to a non-integer positional numeral system that uses the golden ratio as its base. Since this system is positional, convolution can be performed exactly the same as in the binary system.

We complete the algorithm by showing $O(n)$ time procedures for converting between the Zeckendorf system and the golden ratio numeral system.

2 Preliminaries

Fibonacci numbers and Zeckendorf representation. The sequence of Fibonacci numbers is defined as follows (see also Table 1 in the appendix):

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}. \tag{1}$$

Note that this definition works for all integers n (even negative ones). In particular we have

$$F_{-n} = (-1)^{n+1} F_n. \tag{2}$$

We can express non-negative integers as weighted sums of Fibonacci numbers. A sequence of integer weights a_i with only a finite number of non-zero weights represents number

$$x = \sum_i a_i F_i.$$

Zeckendorf [12, 6] showed that under the following conditions such a representation (called *the Zeckendorf representation*) is unique:

(Z1) Each a_i is either 0 or 1.
(Z2) There are no adjacent non-zero weights, thus $a_{i+1}a_i = 0$ for all i.
(Z3) $a_i = 0$ for $i < 2$.

See Table 2 for the Zeckendorf representations of numbers from 1 to 30. Note that we write sequences of weights in such a way that the most significant weight is on the left.

There are several ways of showing that every number has a Zeckendorf representation, but it is fruitful for us to recall a proof from [8]. For a given non-negative integer x we can start from a trivial representation with only one non-zero weight $a_2 = x$, since $F_2 = 1$. This representation violates condition (Z1). The idea is to transform the representation in a series of steps, where every step locally changes weights, but the represented value stays the same. At the end we would like to obtain a representation that satisfies all three conditions. We call such a series of steps a *normalization procedure*.

Following [8] we show here one example of such a procedure, although not efficient (later in Theorem 2 we show an efficient normalization procedure.) We keep an invariant that condition (Z3) is always satisfied, but conditions (Z1) and (Z2) might be not. Every time condition (Z2) is not satisfied, we find the left-most pair of adjacent non-zero weights and apply the following transformation on it:

$$0\bar{x}\bar{y} \to 1xy, \tag{3}$$

where $\bar{x} = x+1$. The soundness of this transformation follows directly from the recurrence (1), and it never leads to violation of condition (Z3).

If only condition (Z1) is not satisfied, we apply the following transformation on the left-most weight greater than 1:

$$0\bar{\bar{x}}0y \to 1x0\bar{y}. \tag{4}$$

The soundness follows from the following equality valid for all integers n:

$$2F_n = F_n + (F_{n-1} + F_{n-2}) = F_{n+1} + F_{n-2}. \tag{5}$$

The transformation may violate condition (Z3), but only when fixing weight a_2, thus it increases weight a_0 then. But this weight is multiplied by $F_0 = 0$ anyway, so we can safely set it back to 0.

Every time we apply transformations (3)–(4), the sum $\sum_i 2^i a_i$ increases, so since every integer has a finite number of representations with non-negative weights, the process terminates.

The uniqueness of Zeckendorf representation follows from a counting argument (see [6]).

The golden ratio numeral system. Binet's formula provides a closed-form solution for Fibonacci numbers and shows a relationship between them and the golden ratio $\varphi = \frac{1+\sqrt{5}}{2}$:

$$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}. \tag{6}$$

In fact, another numeral system closely related to Fibonacci numbers is a positional numeral system using the golden ratio as the base [3, 7]. In this system a sequence of integer weights c_i represents number

$$x = \sum_i c_i \varphi^i.$$

This representation is unique (and we call it *the base-φ representation*; see also Table 2) if it satisfies conditions (Z1) and (Z2). To prove that every non-negative integer has a base-φ representation, observe that φ^i satisfies the same recurrence as Fibonacci numbers F_i, namely

$$\varphi^n = \varphi^{n-1} + \varphi^{n-2}.$$

It means that transformations (3)–(4) also work for base-φ representations, and starting from a trivial representation (with one non-zero weight $c_0 = x$, since $\varphi^0 = 1$) we can use almost the same normalization procedure as before to obtain a representation satisfying conditions (Z1) and (Z2). This time, however, we do not make fixes for condition (Z3), thus non-zero weights can occur also for negative indices.

Addition and subtraction algorithms. Arithmetic operations on Zeckendorf representations were discussed in [5, 4, 10] but authors of these papers either did not analyze time complexity of their algorithms or they provided very weak bounds. The optimal linear-time bounds for addition and subtraction algorithms were given in [1], and we recall them here, since they are needed for the multiplication algorithm presented in this paper.

▶ **Theorem 1.** *There are $O(n)$ algorithms for addition and subtraction of two n-digit numbers in Zeckendorf representation and in base-φ representation.*

Proof. We present here only a method behind the algorithms; the full algorithms with the proofs can be found in [1]. The addition algorithm starts by independently adding weights from corresponding positions in the Zeckendorf (or base-φ) representations, obtaining a sequence of weights from set $\{0, 1, 2\}$. This sequence can violate condition (Z2) by having consecutive 1s and/or condition (Z1) by having 2s (but only adjacent to 0s). Since we need to normalize the sequence, we could apply the transformations (3)–(4), but this would be inefficient. To perform normalization procedure in $O(n)$ we need to be more careful. Let r be the position of the right-most non-zero weight in the sequence.

We move a 4-position-wide window from left to right, applying the following transformations were applicable:

$$020x \to 100\bar{x}, \qquad 030x \to 110\bar{x}, \qquad 021x \to 110x, \qquad 012x \to 101x,$$

where $x \in \{0, 1, 2\}$ and $\bar{x} = x + 1$. These transformations may introduce additional weight 3, but at the end all 3s as well as 2s are removed, restoring condition (Z1). Then we move a 3-position-wide window from left to right and apply transformation $011 \to 100$, which removes groups of consecutive 1s that are longer than two positions. Finally, we move the same window from right to left, removing remaining adjacent 1s and fully restoring condition (Z2).

That finishes normalization procedure for base-φ representations. Note that in the output sequence the right-most non-zero weight is at position not smaller than $r - 2$.

For Zeckendorf representations we must take some additional local step to fix condition (Z3), but its a technical detail.

For subtraction we run a similar algorithm, but now weights are from set $\{-1, 0, 1\}$, and there are no consecutive 1s nor consecutive -1s. We move a 4-position-wide window from left to right with transformations:

$$x00 \to \bar{x}11, \qquad x\bar{1}0 \to \bar{x}01, \qquad x\bar{1}1 \to \bar{x}02, \qquad x0\bar{1} \to \bar{x}10,$$

where $x \in \{1, 2\}$, $\bar{x} = x - 1$ and $\bar{1} = -1$. The transformations keep a positive weight in the window and use it to cancel any -1s. They may introduce additional 2s, but these are adjacent only to 0s. So at the end we can finish the algorithm by performing normalization procedure for addition algorithm. ◀

The subtraction algorithm can be used to prove the uniqueness of the base-φ representations. Suppose that we have two different sequences of weights c_i and c'_i that satisfy conditions (Z1) and (Z2), and they represent the same value x. Assume that c_i is lexicographically larger than c'_i. If we subtract c'_i from c_i, we get a non-zero sequence of weights satisfying conditions (Z1)–(Z2) that represents value 0. That is impossible, since all φ^i are positive.

3 Multiplication Algorithms

Several multiplication algorithms for Zeckendorf representation were discussed in [1], but the authors did not find any algorithm with time complexity better than $O(n^2)$. They posed this as a "challenging open problem".

The idea of the algorithm presented in this paper is that since base-φ is a positional numeral system, the multiplication in this system can be reduced to a convolution after which we need to do a normalization phase. Such a convolution can be calculated in $O(n \log n)$ time using the Fast Fourier Transform [9].

Let sequences c_j and c'_j be the base-φ representations of numbers x and x'. As a result of the convolution we obtain a sequence of non-negative weights C_i such that

$$C_i = \sum_j c_j c'_{i-j},$$

and the sum $\sum_i C_i \varphi^i$ is equal to the product $x \cdot x'$. The weights C_i can be up to $O(n)$. Linear-time normalization procedures used in addition and subtraction algorithms utilised the fact that weights were of constant size. However, we can devise an efficient normalization procedure for arbitrary weights:

▶ **Theorem 2.** *There is $O(n \log M)$ algorithm that, given a sequence of n integer weights from range $[0, M]$, normalizes it so it satisfy conditions (Z1) and (Z2).*

Proof. Each weight is a number of $m = \lfloor \log M \rfloor + 1$ bits. We create m binary sequences $x_0, x_1, \ldots, x_{m-1}$: each number in sequence x_i equals to the i-th bit of the corresponding number in the original sequence. We initialize the answer to be a sequence of 0s.

Then we perform m phases. In phase i (for $i = m-1, \ldots, 1, 0$) we multiply the answer by 2 (by executing addition algorithm in $O(n)$ time) and then we add to the answer sequence x_i (again by executing addition algorithm). ◀

Using the above theorem directly, we get a normalization procedure for base-φ representation that works in $O(n \log n)$ time and completes the multiplication algorithm for the golden ratio system.

The idea for a multiplication algorithm for Zeckendorf representation is to convert both numbers into base-φ representation, perform multiplication in base-φ, and then convert the result back to Zeckendorf representation. In the next section we show that such conversions are possible in $O(n)$ time, thus we get the theorem:

▶ **Theorem 3.** *There are $O(n \log n)$ algorithms for multiplication of two n-digit numbers in Zeckendorf representation and in base-φ representation.*

4 Conversions Between Representations

Compare the proofs that every number x has the Zeckendorf and the base-φ representation. In both of them we started with a trivial representation ($a_2 = x$ and $c_0 = x$, respectively), and we performed a certain normalization procedure.

Now imagine that we start from $\alpha_2 = x$ (like in the proof for Zeckendorf representation) and we perform the normalization procedure that ignores condition (Z3) (like in the proof for base-φ representation). We get a sequence of weights α_i that satisfies conditions (Z1)–(Z2) and represents number $x = \sum_i \alpha_i F_i$. But exactly the same procedure applied to initial condition $c_0 = x$ produces the base-φ representation $x = \sum_i c_i \varphi^i$. Therefore $\alpha_i = c_{i-2}$ for all integers i.

Thus the weights of the base-φ representation shifted by two places and applied to Fibonacci numbers yield the same value:

$$\sum_i c_i \varphi^i = x = \sum_i c_{i-2} F_i.$$

Thanks to that property conversion from base-φ to Zeckendorf is easy:

▶ **Theorem 4.** *There is $O(n)$ algorithm for converting an n-digit number in base-φ representation to Zeckendorf representation.*

Proof. We have the base-φ representation $x = \sum_i c_i \varphi^i$. For easier notation denote $\alpha_i = c_{i-2}$. We create four sequences that partition the weights into four parts: for positive indices, for indices close to the origin, for odd negative indices, and for even negative indices:

$$x_{pos} = \sum_{i \geq 2} \alpha_i F_i, \qquad x_{odd} = \sum_{i \geq 2,\ i \text{ odd}} \alpha_{-i} F_i,$$

$$x_{orig} = \alpha_1 \alpha_{-1} F_3 + [\alpha_1 \neq \alpha_{-1}] F_2, \qquad x_{even} = \sum_{i \geq 2,\ i \text{ even}} \alpha_{-i} F_i.$$

Since sequence α_i satisfies conditions (Z1)–(Z2), the above sequences are the Zeckendorf representations of numbers x_{pos}, x_{orig}, x_{odd}, and x_{even}. Moreover, from (2) we know that Fibonacci numbers of negative indices $-i$ are equal to their positive counterparts but with changed sign if i is even, thus $x = x_{pos} + x_{orig} + x_{odd} - x_{even}$. Thus we can calculate the Zeckendorf representation of x using the addition algorithm twice and the subtraction algorithm once. ◀

The representation α_i has been investigated by Zeckendorf who called it a "generalized Fibonacci numeration" [11].

For conversion from Zeckendorf to base-φ we can look at Table 2 that contains the base-φ representations for small integers and observe that certain representations have simple forms, e.g. there are some that contain only two 1s. Moreover, if we allow weights of -1 (we call it *relaxed* base-φ representation) we can get simple representations for some more numbers (see Table 3).

These numbers that happen to have simple forms are called Lucas numbers L_n, and are defined as follows (see also Table 1):

$$L_n = F_{n+1} + F_{n-1}.$$

In particular we have the following relationship between Lucas numbers and the golden ratio, that is easy to check using Binet's formula:

$$L_n = \varphi^n + (-1)^n \varphi^{-n}. \tag{7}$$

Thus it looks like, as Fibonacci numbers were the simplest building blocks of Zeckendorf representations, we could use Lucas numbers as simple building blocks of relaxed base-φ representations. Thus we can try to express integers using weighted sums of Lucas numbers; a sequence of integer weights b_i represents number

$$x = \sum_i b_i L_i.$$

Each non-negative integer can be represented as a weighted sum of Lucas numbers, and this representation is unique (thus we call it *the Lucas representation*, see [6] and Table 2) if it satisfies conditions (Z1), (Z2) and a slightly modified version of condition (Z3):
(Z3′) $a_i = 0$ for $i < 0$, and $a_2 a_0 = 0$.

Such a representation can be obtained almost greedily from the Zeckendorf representation:

▶ **Theorem 5.** *There is $O(n)$ algorithm for converting an n-digit number in Zeckendorf representation to Lucas representation.*

Proof. We have the Zeckendorf representation $x = \sum_i a_i F_i$. Like in the proof of Theorem 1, we move a 4-position-wide window from left to right over sequence a_i and apply the following transformations were applicable:

$$100x \to 011x, \qquad 101x \to 0\bar{0}0x, \qquad 1100 \to 0\bar{0}01, \qquad 1101 \to 01\bar{1}0, \qquad 1110 \to 0\bar{1}00,$$

where $x \in \{0, 1\}$. A bar over a digit at position i means that we must increment weight b_i by one. This way we are zeroing sequence a_i, and at the same time constructing the Lucas representation b_i, keeping the sum $\sum_i (a_i F_i + b_i L_i)$ unchanged. We keep the invariant that all the time sequence a_i satisfies conditions (Z1)–(Z2), except the left-most group of consecutive 1s, which can be of length up to 3. Thanks to that every transformation applied replaces the left-most 1 with 0.

We stop after the transformation over positions 3 through 0. Then we finish by applying one of the following transformations over positions 2 through 0:

$$110 \to 00\bar{0}, \qquad 100 \to 0\bar{0}0, \qquad 010 \to 0\bar{0}0, \qquad 001 \to 000.$$

Each weight b_i can be incremented at most twice (if it is incremented twice, then one increment is due to transformation $1101 \to 01\bar{1}0$ followed by transformation $1100 \to 0\bar{0}01$). Moreover, careful examination of the transformations shows that no two adjacent weights are incremented. Also, there is at most one increment over positions 2 through 0, thus the last three weights are in set $\{000, 001, 010, 100\}$.

Thus we have sequence b_i of weights from set $\{0, 1, 2\}$, and 2s are adjacent to 0s on both sides. Since Lucas numbers also satisfy recurrence $L_n = L_{n-1} + L_{n-2}$, we can perform the same normalization procedure as in addition algorithm for base-φ representation, which would help us restore conditions (Z1) and (Z2). However, this procedure could introduce some non-zero weights b_{r-1} and b_{r-2}, where r is the index of the right-most non-zero weight in b_i; so we must be careful here.

If the last three weights are 000 or 100, we are safe, since $r \geq 2$. For the last three weights being 001 or 010 we decrement weight b_r to zero, perform the normalization procedure, and increment b_r back.

For 001 case we end up either with 0101 (which is safe) or with 011 (which we replace with 100 and perform the normalization once again).

For 010 case we end up either with 0110 (which we replace with 1000 and perform the normalization once again) or with 020 (which we replace with 001).

This restores conditions (Z1) and (Z2). If condition (Z3′) is not satisfied, the representation ends in $x0101$. We can replace it with $x1010$ and if $x = 1$ we run a 3-position-wide window from right to left with transformation $011 \to 100$. ◀

Finally, we show conversion from Lucas to base-φ. Conversion from Zeckendorf to base-φ goes through an intermediate step of the Lucas representation.

▶ **Theorem 6.** *There is $O(n)$ algorithm for converting an n-digit number in Lucas representation to base-φ representation.*

Proof. From the Lucas representation $x = \sum_i b_i L_i$ we can directly obtain a sequence of weights c_i in a relaxed base-φ representation, by using (7):

$$c_i = \begin{cases} b_i & i > 0, \\ 2b_0 & i = 0, \\ (-1)^i b_{-i} & i < 0. \end{cases}$$

If $c_0 = 0$, all weights are from set $\{-1, 0, 1\}$, so we can normalize this sequence by performing the normalization procedure just like in subtraction algorithm for base-φ representation. Otherwise $c_0 = 2$, and first we zero this weight, perform the subtraction normalization, then increment c_1 and c_{-2} by one and perform the normalization procedure like in addition algorithm. ◀

5 Alternative Explanation

It turns out that we can invent the multiplication algorithm presented in this paper without using terminology of the golden ratio numeral system, and staying only in the realm of Fibonacci numbers. In fact this was the way it was invented by the author of this paper. Since the obtained results are the same, and we simply change the language used, this section present only the essence of this approach.

The idea is to begin with the observation that starting from trivial representation $\alpha_2 = x$ and performing normalization procedure that ignores condition (Z3) gives us a sequence α_i satisfying conditions (Z1)–(Z2) and representing number

$$x = \sum_i \alpha_i F_i.$$

But if we start exactly the same procedure from an initial condition $\alpha'_m = x$, which represents number xF_m, we get an equation

$$xF_m = \sum_i \alpha'_i F_i = \sum_i \alpha_{i+2} F_{i+m}. \tag{8}$$

The equation captures an important property that shifting sequence α_i results in multiplying the represented value by consecutive Fibonacci numbers. In positional systems (like the binary or the golden ratio system) an analogous property states that shifting a sequence of digits results in multiplying the represented value by the base (2 or φ, respectively).

This "shiftable" property is enough for convolution to work:

$$x \cdot x' = x\Big(\sum_i a'_i F_i\Big) = \sum_i a'_i x F_i \overset{(8)}{=} \sum_i a'_i \Big(\sum_j \alpha_{j+2} F_{j+i}\Big) = \sum_i \Big(\sum_j a'_{i-j} \alpha_{j+2}\Big) F_i.$$

Thus it is enough to convert one of the arguments to a "shiftable" representation α_i, and perform convolution to obtain weights $A_i = \sum_j a'_{i-j}\alpha_{j+2}$ of a "shiftable" representation of the product.

For conversion part we observe that the representations α_i of Lucas numbers are particularly simple, which results from the following equivalent of equation (7) that holds for any integers n and m:

$$L_n F_m = F_{m+n} + (-1)^n F_{m-n}.$$

References

1 Connor Ahlbach, Jeremy Usatine, Christiane Frougny, and Nicholas Pippenger. Efficient algorithms for Zeckendorf arithmetic. *The Fibonacci Quarterly*, 51(3):249–255, 2013.

2 Alberto Apostolico and Aviezri S. Fraenkel. Robust transmission of unbounded strings using Fibonacci representations. *IEEE Transactions on Information Theory*, 33(2):238–245, 1987.

3 George Bergman. A number system with an irrational base. *Mathematics Magazine*, 31(2):98–110, 1957.

4 Peter Fenwick. Zeckendorf integer arithmetic. *Fibonacci Quarterly*, 41:405–413, 2003.

5 H.T. Freitag and G.M. Phillips. *Elements of Zeckendorf Arithmetic*, pages 129–132. Springer Netherlands, Dordrecht, 1998.

6 Verner E. Hoggatt, Jr. *Fibonacci and Lucas Numbers.* Houghton Mifflin Company, 1969.

7 Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison–Wesley, 1968.

8 Donald E. Knuth. Fibonacci multiplication. *Applied Mathematics Letters*, 1(2):III–VI, 1988.

9 Arnold Schönhage and Volker Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

10 Garry J. Tee. Russian peasant multiplication and Egyptian division in Zeckendorf arithmetic. *Australian Mathematical Society Gazette*, 30(5):267–276, 2003.

11 Edouard Zeckendorf. A generalized Fibonacci numeration. *The Fibonacci Quarterly*, 10(4):365–372, 1972.

12 Edouard Zeckendorf. Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas. *Bull. Soc. Roy. Sci. Liège*, 41:179–182, 1972.

Table 1 Fibonacci and Lucas numbers.

n	8	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
F_n	21	13	8	5	3	2	1	1	0	1	-1	2	-3	5	-8	13	-21
L_n	47	29	18	11	7	4	3	1	2								

Table 2 Zeckendorf, base-φ, and Lucas representations.

n	Zeckendorf	base-φ	Lucas
1	100	1.	10
2	1000	10.01	1
3	10000	100.01	100
4	10100	101.01	1000
5	100000	1000.1001	1010
6	100100	1010.0001	1001
7	101000	10000.0001	10000
8	1000000	10001.0001	10010
9	1000100	10010.0101	10001
10	1001000	10100.0101	10100
11	1010000	10101.0101	100000
12	1010100	100000.101001	100010
13	10000000	100010.001001	100001
14	10000100	100100.001001	100100
15	10001000	100101.001001	101000
16	10010000	101000.100001	101010
17	10010100	101010.000001	101001
18	10100000	1000000.000001	1000000
19	10100100	1000001.000001	1000010
20	10101000	1000010.010001	1000001
21	100000000	1000100.010001	1000100
22	100000100	1000101.010001	1001000
23	100001000	1001000.100101	1001010
24	100010000	1001010.000101	1001001
25	100010100	1010000.000101	1010000
26	100100000	1010001.000101	1010010
27	100100100	1010010.010101	1010001
28	100101000	1010100.010101	1010100
29	101000000	1010101.010101	10000000
30	101000100	10000000.10101001	10000010

Table 3 Base-φ representations of Lucas numbers and their simple forms in relaxed base-φ.

n	base-φ	relaxed base-φ
1	1.	$10.\bar{1}$
3	100.01	100.01
4	101.01	$1000.00\bar{1}$
7	10000.0001	10000.0001
11	10101.0101	$100000.0000\bar{1}$
18	1000000.000001	1000000.000001
29	1010101.010101	$10000000.000000\bar{1}$