Fibonacci Facts and Formulas

R.M. Capocelli
Dipartimento di Informatica ed Applicazioni
Universita' di Salerno
Salerno, Italy 84100

G. Cerbone
P. Cull
J.L. Holloway
Department of Computer Science
Oregon State University
Corvallis, Oregon (USA) 97331-3902

88-50-4

Fibonacci Facts and Formulas

R. M. Capocelli*
G. Cerbone†
P. Cull
J. L. Holloway
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

Abstract

We investigate several methods of computing Fibonacci numbers quickly and generalize some properties of the Fibonacci numbers to degree r Fibonacci (R-nacci) numbers. Sections 2 and 3 present several algorithms for computing the traditional, degree two, Fibonacci numbers quickly. Sections 4 and 5 investigate the structure of the binary representation of the Fibonacci numbers. Section 6 shows how the generalized Fibonacci numbers can be expressed as rounded powers of the dominant root of the characteristic equation. Properties of the roots of the characteristic equation of the generalized Fibonacci numbers are presented in Section 7. Section 8 introduces several properties of the Zeckendorf representation of the integers. Finally, in Section 9 the asymptotic proportion of 1's in the Zeckendorf representation of integers is computed and an easy to compute closed formula is given.

1 Introduction

We define the Fibonacci numbers of degree r, R-nacci numbers, as:

$$F_0^{(r)} = 0,\ F_1^{(r)} = 1,\ F_j^{(r)} = 2^{j-2} \text{ for } j = 2, 3, 4, \ldots, r-1;$$
$$F_i^{(r)} = F_{i-1}^{(r)} + F_{i-2}^{(r)} + F_{i-3}^{(r)} + \cdots + F_{i-r+1}^{(r)} + F_{i-r}^{(r)},\ i \geq r. \quad (1)$$

The Fibonacci, Tribonacci [6], and Quadranacci [8] numbers arise as special cases of equation (1) by letting $r = 2$, $r = 3$, and $r = 4$, respectively. The terms Fibonacci numbers and F_n will refer to degree two Fibonacci numbers unless specified otherwise.

---

*On leave from Dipartimento di Informatica ed Applicazioni. Universita' di Salerno, Salerno, Italy 84100

†Supported by a fellowship from the Italian National Research Council (CNR 203.01.43)

The roots of the characteristic equation associated with the R–nacci difference equation will be indicated by $\lambda_i^{(r)}$ $(1 \leq i \leq r)$. In particular, $\lambda_1 = \lambda_1^{(2)} = \frac{1+\sqrt{5}}{2}$ and $\lambda_2 = \lambda_2^{(2)} = \frac{1-\sqrt{5}}{2}$ are the roots of the characteristic polynomial for the Fibonacci numbers.

The Lucas numbers are defined to be the sequence of numbers

$$l_n = l_{n-1} + l_{n-2},\ n \geq 2,\ l_0 = 2,\ l_1 = 1. \tag{2}$$

These numbers are a solution of the Fibonacci difference equation which is linearly independent of the Fibonacci numbers. As we will see, the Lucas numbers can be used to compute the Fibonacci numbers.

2 Algorithms

The previous work on algorithms to compute the Fibonacci numbers used the straight line code model with the uniform cost function of computation [1] to measure time complexity [5,7,10,12,14]. As the operands become large, the uniform cost assumption is no longer valid and the time required to operate on large operands must be taken into account. Since F_n grows exponentially with n, the uniform cost function is unsuitable as a measure of the time used to compute the Fibonacci numbers. In our work, the bitwise model is used to analyze and compare the various algorithms that compute Fibonacci numbers since it reflects the cost of operating on variable size operands [1].

Most of our algorithms use multiplication. We will report the number of bit operations assuming that we are using a multiplication which uses n^2 bit operations to multiply two n bit numbers. There are, of course, multiplication algorithms which use fewer than n^2 bit operations (at least for large n). Use of one of these faster algorithms would give the same ordering on our algorithms which use multiplication, and would make all of these algorithms, including Binet's formula, faster than repeated addition. But we want to point out that on one hand without a faster multiplication, repeated addition is faster than the computation from Binet's formula, while on the other hand we can create algorithms that are faster than repeated addition even without faster multiply.

We will use γn to represent the number of bits in F_n where $\gamma = \log \lambda_1 \approx .69424$, because F_n is asymptotic to $\frac{\lambda^n}{\sqrt{5}}$.

2.1 Repeated addition

A common algorithm used to find F_n computes the sequence $F_0, F_1, F_2, \ldots, F_n$ by starting with $F_0 = 0$, $F_1 = 1$ and computing each successive F_i, $2 \leq i \leq n$, by adding F_{i-2} and F_{i-1}. The number of bit operations used to compute F_n using the repeated addition algorithm is $\sum_{i=1}^{n-1} \gamma i = \gamma \frac{n(n-1)}{2}$.

fib (n)

if n $= 1$ or n $= 2$ return 1

else return $F_n = \left\lceil (\text{fib}(n/2))^2 \cdot \sqrt{5} \right\rceil$

Figure 1: Recursive Binet's approximation

2.2 Binet's formula

Binet's closed form formula for F_n is:

$$F_n = \frac{1}{\sqrt{5}}(\lambda_1^n - \lambda_2^n) \tag{3}$$

and $\lambda_2 \approx -0.61803$ so $|\lambda_2^n|$ gets small as n increases. Furthermore Knuth [9] shows

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n + 0.5 \right\rfloor, \; n \geq 0. \tag{4}$$

Using $\Theta(n^2)$ multiply, this algorithm uses $\gamma^2 n^2 \log n$ bit operations to compute F_n. A recursive function to compute F_n based on Binet's formula is given in Figure 1. To see that this algorithm correctly computes F_n, $n = 2^k$, $k \geq 0$, we have

$$\begin{aligned} F_n &= \frac{1}{\sqrt{5}}(\lambda_1^n - \lambda_2^n) \\ \sqrt{5}\,(F_n)^2 &= \frac{1}{\sqrt{5}}\left(\lambda_1^{2n} - 2(-1)^n + \lambda_2^{2n}\right) \end{aligned}$$

and

$$F_{2n} = \frac{1}{\sqrt{5}}\left(\lambda_1^{2n} - \lambda_2^{2n}\right).$$

Subtracting, we get

$$\epsilon = F_{2n} - \sqrt{5}(F_n)^2 = \frac{2}{\sqrt{5}}\left(-\lambda_2^{2n} + (-1)^n\right).$$

Since $0 < \epsilon < 1$,

$$F_{2n} = \sqrt{5}(F_n)^2 + \epsilon = \left\lceil \sqrt{5}(F_n)^2 \right\rceil.$$

The number of bit operations used by this algorithm is $\frac{5}{3}\gamma^2 n^2$.

fib (n)

if $n = 1$ return $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

else

$\begin{pmatrix} F_{n/2+1} & F_{n/2} \\ F_{n/2} & F_{n/2-1} \end{pmatrix} \leftarrow$ fib $(n/2)$

$F_{n+2} \leftarrow F_{n/2+1}(F_{n/2+1} + 2F_{n/2})$

$F_n \leftarrow F_{n/2}(2F_{n/2+1} - F_{n/2})$

return $\begin{pmatrix} F_{n+2} - F_n & F_n \\ F_n & F_{n+2} - 2F_n \end{pmatrix}$

Figure 2: Two multiply matrix algorithm

2.3 Matrix Algorithms

The matrix identity

$$A^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

can be used to compute Fibonacci numbers.

By using repeated squarings, F_n can be computed using $\log n$ matrix multiplies or $\frac{8}{3}\gamma^2 n^2$ bit operations using the standard matrix multiplication algorithm.

2.3.1 Two Full Size Multiplies

A^{2n} can be computed from A^n using only two n-bit multiplications. Vorob'ev [15] proves

$$F_{n+k} = F_{n-1}F_k + F_nF_{k+1} \tag{5}$$

leading to

$$\begin{aligned} F_{2n} &= F_{n-1}F_n + F_nF_{n+1} = F_n(2F_{n+1} - F_n) \\ F_{2n+1} &= F_{n-1}F_{n+1} + F_nF_{n+2} \\ F_{2n+2} &= F_nF_{n+1} + F_{n+1}F_{n+2} = F_{n+1}(F_{n+1} + 2F_n) \end{aligned} \tag{6}$$

Using the equations for F_{2n} and F_{2n+2}, A^{2n} can be computed from A^n using only two multiplications. The algorithm is given in Figure 2 and uses $\frac{2}{3}\gamma^2 n^2$ bit operations.

2.3.2 One Full Size and Two Half Size Multiplies

We can compute F_n using one full size and two half size multiplies instead of two full size multiplications by noticing that $F_n = F_{n/2}(2F_{n/2+1} - F_{n/2})$ and $F_{n/2+1} = F_{n/4}^2 + F_{n/4+1}^2$. Two half size multiplies are avoided since two of the values used in computing $F_{n/2}$ are stored and used to compute $F_{n/2+1}$. The algorithm is given in Figure 3 and uses $\frac{1}{2}\gamma^2 n^2$ bit operations.

fib (n)

if n = 4 return $F_2 = 1,\ F_3 = 2,\ F_4 = 3$

else

$F_{n/4}, F_{n/4+1}, F_{n/2} \leftarrow$ fib $(n/2)$

$F_{n/2+1} \leftarrow F_{n/4+1}^2 + F_{n/4}^2$

$F_n \leftarrow F_n = F_{n/2}(2F_{n/2+1} - F_{n/2})$

return $F_{n/2},\ F_{n/2+1},\ F_n$

Figure 3: **1 + 2/2 multiply matrix algorithm**

2.4 Product of Lucas Numbers

The Lucas number of the form $l_{2^i}, i \geq 0$, can be used to compute Fibonacci numbers. We can compute $F_{2^i},\ i \geq 1$, by

$$F_{2^i} = \prod_{k=1}^{i-1} l_{2^k}.$$

Since $l_{2^i} = (l_{2^{i-1}})^2 - 2,\ i \geq 2$, and $\frac{F_{2n}}{F_n} = l_n$, we get the relation

$$\frac{F_{4n}}{F_{2n}} = \left(\frac{F_{2n}}{F_n}\right)^2 - 2(-1)^n. \tag{7}$$

Equation (7) can be used to compute Fibonacci numbers using $\frac{5}{12}\gamma^2 n^2$ bit operations.

The sequence $F_{4+k}, F_{8+k}, F_{16+k}, \ldots, F_{2^{i+1}+k}$ can be computed for any integer k using the relation $F_{2^i+k} = F_{(2^{i-1}+k)}l_{2^i} - F_k$.

The algorithm presented in Figure 4 computes the sequence of Lucas numbers and uses this sequence to compute the two sequences of Fibonacci numbers, $F_1, F_3, F_7, \ldots F_{2^{sl}-1}$ and $F_2, F_4, F_8, \ldots F_{2^{sl}}$. Fib-help, Figure 5, is a function that will compute F_n for any n when given these two Fibonacci sequences. Equation (5) tells

fib (n)

sl $\leftarrow \lceil \log n \rceil - 1$

Lucas-sequence $\leftarrow l_1, l_2, l_4, \ldots, l_{2^{sl-1}}$

f1-sequence $\leftarrow F_1, F_3, F_7, \ldots, F_{2^{sl}-1}$

f2-sequence $\leftarrow F_2, F_4, F_8, \ldots, F_{2^{sl}}$

fib-help (n sl f1-sequence f2-sequence)

Figure 4: **Product of Lucas numbers algorithm to compute any F_n, part 1**

fib-help (n expo f1seq f2seq)

if n = 0 return 0

else if n = 1 return 1

else if n = 2 return 1

else return

f1seq[expo] * fib-help ($n - 2^{expo}$, $\lceil \log(n - 2^{expo}) \rceil - 1$, f1seq, f2seq) +

f2seq[expo] * fib-help ($n - 2^{expo} \div 1$, $\lceil \log(n - 2^{expo}) \rceil - 1$, f1seq, f2seq)

Figure 5: **Product of Lucas numbers algorithm to compute any F_n, part 2**

	Addition		Binet		$1 + 2/2$ Matrix		Product of Lucas Numbers	
n	act	comp	act	comp	act	comp	act	comp
2^8	0.22	0.28	14.83	3.09	0.03	0.00	0.02	0.00
2^9	0.62	1.10	11.10	13.92	0.03	0.01	0.02	0.00
2^{10}	5.17	4.42	77.8	61.86	0.07	0.02	0.05	0.02
2^{11}	16.73	17.71	287.7	272.16	0.18	0.08	0.12	0.07
2^{12}	71.30	70.84	1187.7		0.47	0.33	0.37	0.27
2^{13}	279.5	283.39			1.58	1.33	1.23	1.10
2^{14}	1133.6				9.23	5.32	4.60	4.38
2^{15}					21.60	21.29	17.80	17.53
2^{16}					84.98	85.17	74.00	70.14
2^{17}					340.7		280.2	280.5
2^{18}							1121.9	
constant	4.223 E-6		5.899 E-6		1.983 E-8		1.633 E-8	
bit ops	$\gamma^{\frac{(n-1)n}{2}}$		$\gamma^2 n^2 \log n$		$\frac{1}{2}\gamma^2 n^2$		$\frac{5}{12}\gamma^2 n^2$	

Table 1: Running Time (CPU seconds)

us that to compute F_{2^i+k} we need four numbers: F_{2^i-1}, F_{2^i}, F_k, and F_{k+1}. We already know F_{2^i-1} and F_{2^i} and we can compute F_k and F_{k+1} by recursively calling fib-help.

3 Running Times

The number of bit operations used by each algorithm is an accurate predictor of the running time of that algorithm. The time used by an algorithm to compute a large Fibonacci number was used to compute a constant and that constant was used to predict the time required to compute other Fibonacci numbers. For some large n, the constants were computed by dividing the time used to compute F_n by n^2 ($n^2 \log n$ for Binet's formula). The algorithms were implemented in Ibuki Common Lisp (IBCL) on a Sequent Balance 21000. The actual running times, computed running times, constants, and number of bit operations used are presented in Table 1 and confirm empirically that the fastest algorithm we have found to compute F_n is the product of Lucas numbers algorithm using $\frac{5}{12}\gamma^2 n^2$ bit operations.

4 Compression and Randomness

Three compression algorithms, Ziv-Lempel [18], UNIX[1] compress using the Lempel-Ziv-Welch [16] algorithm, and UNIX compact using the dynamic Huffman compres-

[1]UNIX is a trademark of Bell Laboratories.

	Sequence			
Method	Integer	Powers	Fibonacci	Random
Z–L	1.41	0.42	1.89	1.89
Unix compress	1.39	0.03	1.47	1.47
Unix compact	0.97	0.13	1.02	1.02

Table 2: Compression Ratios for Various Sequences

sion algorithm were used to compress the first 2^{14} bits of four sequences. The sequences were formed by concatenating the binary representation of integers from an integer sequence where the integer sequences are the Fibonacci numbers $F_0, F_1, F_2 \ldots$, the positive integers $1, 2, 3, \ldots$, the powers of two $2^0, 2^1, 2^2, \ldots$, and the integers generated by the linear congruential random number generator in IBCL. For example, the binary sequence for 1 2 3 4 5 ... would be 11011100101 .... The ratio of the size of the compacted sequence to the size of the original sequence is given in Table 2. All three compression algorithms failed to compress the Fibonacci sequence. The compression ratio of the Fibonacci bit sequence and the random bit sequence were similar and led us to test the Fibonacci bit sequence as a random sequence. These results are not unexpected since the compression algorithms assume a finite state source, while the Fibonacci numbers come from a growing source. The compression results are currently only empirical and further investigation is needed.

The Fibonacci bit sequence was submitted to three tests of randomness: the equidistribution test, the serial test [9], and the Wald–Wolfowitz test for total number of runs [11]. The pseudo–random bits from IBCL's random number generator were submitted to the same tests. Both sequences passed the tests of randomness with similar result on each test. It appears that the Fibonacci sequence may be a cheap source of pseudo–random numbers since the next n bits of the sequence can be computed using only n bit operations.

5 Cycles in the i^{th} Bit of $F_n^{(r)}$

The i^{th} bit of the degree r Fibonacci numbers has period $(r+1) \cdot 2^i$. This is proved by showing $L^{(r+1)\cdot 2^n} \bmod 2^{n+1} = I$, where L is the companion matrix for the degree r Fibonacci numbers and I is the identity matrix. These cycles can be used to compute the low order $\log n$ bits of F_n quickly. The cycles in bits 0 and 1 of the Fibonacci sequence can be seen in Table 3. More complex patterns exist and we hope to be able to use these patterns to compute the higher order bits of F_n quickly.

bit	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}
b_0	0	1	1	0	1	1	0	1	1	0	1	1
b_1	0	0	0	1	1	0	0	0	0	1	1	0
b_2	0	0	0	0	0	1	0	1	1	0	1	0

Table 3: Cycles in the b^{th} bit of F_n

6 $F_n^{(r)}$ are Rounded Powers

It is well known that $F_n = \left\lfloor \frac{1}{\sqrt{5}}(\lambda_1)^n + 0.5 \right\rfloor$, $n \geq 0$, and Spickerman [13] has recently shown that degree three Fibonacci numbers are rounded powers of the unique positive real root of the characteristic polynomial for degree three Fibonacci numbers. We have generalized these results and proved that for all $r \geq 2$

$$F_n^{(r)} = \left\lfloor \frac{\lambda_1^{(r)} - 1}{(r+1)\lambda_1^{(r)} - 2r} \left(\lambda_1^{(r)}\right)^{n-1} + .5 \right\rfloor, \; n \geq 0, \tag{8}$$

where $\lambda_1^{(r)}$ is the unique positive real root of the polynomial $x^r - x^{r-1} - \cdots - x - 1$, which is the characteristic polynomial for Fibonacci numbers of degree r.

Proof outline[3]

1) $F_{n+1} = 2F_n - F_{n-r}$
2) $F_n = \sum b_i \left(\lambda_i^{(r)}\right)^n$, $d_n = F_n - b_1 \left(\lambda_1^{(r)}\right)^n$
3) $d_{n+1} = 2d_n - d_{n-r}$
4) At most $r-1$ consecutive $d's$ have the same sign
5) If $|d_n| \geq \alpha$ then for some i, $r \geq i \geq 2$, $|d_{n-i}| > \alpha$
6) $\max\{|d_{-r+2}|, |d_{-r+1}|, \ldots, |d_0|, |d_1|\} < \frac{1}{2}$
7) $b_1 = \frac{\lambda_1^{(r)} - 1}{\lambda_1^{(r)}\left[(r+1)\lambda_1^{(r)} - 2r\right]}$

7 Properties of the roots

In the previous section we showed how to compute R-nacci numbers by using only the dominant root and neglecting the contribution of all other roots. In this section we list, without proof, some of the properties of the roots.

We recall that the characteristic equation associated with R-nacci numbers is: $f(x) = x^r - x^{r-1} - \cdots - x - 1 = 0$ whose roots are called $\lambda_i^{(r)}$. Let $z = a + ib$ be a complex number, we denote a as $Re(z)$, $\sqrt{a^2 + b^2}$ as $\mid z \mid$, and the angle of z in polar coordinates as $arg(z)$.

R1. The characteristic equation has r distinct roots one of which, called *dominant*, is positive and lies between 1 and 2.

R2. The dominant roots of the characteristic equations for different r's form a strictly increasing succession:

$$1.618 < \lambda_1^{(2)} < \lambda_1^{(3)} < \ldots < \lambda_1^{(r)} < \ldots < 2$$

R3. For every $r \geq 2$:

$$2 - \frac{2}{2^r} < \lambda_1^{(r)} < 2 - \frac{1}{2^r}.$$

Note that this is weaker than R2 but gives an immediate range for the value of the dominant root given r.

R4. The $r-1$ nondominant roots have absoute values less than 1 and as r increases these roots approach the $r-1$ nonpositive r^{th} roots of unity.

R5. This rule gives a bound on the modulus of the non dominant roots. For every $1 \leq i \leq r$

$$\frac{1}{\sqrt[r]{3}} < \mid \lambda_i^{(r)} \mid < 1$$

For R6, R7, and R8 we shall call $\lambda_i^{(r)}$ with $1 \leq i \leq \lfloor \frac{r-1}{2} \rfloor$ the distinct and non conjugate roots of the characteristic equation.

R6. This rule gives a bound on the argument of the complex roots. For each $1 \leq i \leq \frac{r-1}{2}$, the argument of complex roots $\lambda_i^{(r)}$, satisfies

$$\frac{2i\pi}{r} < arg\left[\lambda_i^{(r)}\right] \leq \frac{2i\pi}{r-1}.$$

R7. This rule computes the sum of the real part of the complex roots.

$$-\frac{1}{2} < \sum_{j=1}^{\frac{r-1}{2}} Re(\lambda_j^{(r)}) = \frac{1-\lambda_1^{(r)}}{2} < 0$$

R8. This rule computes the product of the moduli of the complex roots:

$$\prod_{j=1}^{\frac{r-1}{2}} \mid \lambda_j^{(r)} \mid^2 = (\lambda_1^{(r)})^{-1}$$

8 Zeckendorf representation of integers

Recently the Zeckendorf representation of integers has been shown to be a useful alternative to the binary representation. In this section, after the initial definitions, we shall list some of the properties of the Zeckendorf representation and show that it requires more space than the binary representation. The space being measured in the number of bits needed to encode a given set of characters. However, we shall also

prove that for a large enough degree ($r = 6$) the two representations "practically" require the same amount of space.

Each nonnegative integer N has the following unique Zeckendorf representation in terms of Fibonacci numbers of degree r [8,17].

$$N = a_2 F_2^{(r)} + a_3 F_3^{(r)} + a_4 F_4^{(r)} + ... + a_j F_j^{(r)}$$

where $a_i \in \{0,1\}$ and $a_i a_{i-1} a_{i-2} a_{i-3} \ldots a_{i-r+1} = 0$ (no r consecutive a's are 1).

Like the binary representation of integers, the Zeckendorf representation can be written as a string of 0's and 1's; i.e. $a_j a_{j-1} a_{j-2} ..... a_3 a_2$ (where leading zeros are ineffective).

Capocelli [2] gives an efficient algorithm to derive the Zeckendorf representation of integers. In the following, we list without proof some relevant properties of the Zeckendorf representation.

Z1. The number of Zeckendorf sequences of length $n-1$ is $F_{n+1}^{(r)}$.

Z2. The number of Zeckendorf sequences of length $n-1$ that begin with the symbol 1 is $F_{n+1}^{(r)} - F_n^{(r)}$.

Z3. The Zeckendorf representation of $F_{n+1}^{(r)}$ is $1 \underbrace{00\ldots0}_{n-1}$.

Z4. The Zeckendorf representation of $\sum_{i=1}^{h} F_{n+i}^{(r)}$ is $\underbrace{11\ldots1}_{h}\underbrace{00\ldots0}_{n-1}$ $h < r$.

Z5. Let t and j be integers, $t > 0$ and $F_{n-t}^{(r)} \leq j < F_{n-t+1}^{(r)}$, and let α be the Zeckendorf representation of j. The Zeckendorf representation of $j + \sum_{i=1}^{h} F_{n+i}^{(r)}$, $h < r$, is $\underbrace{11\ldots1}_{h}\underbrace{00\ldots0}_{t}\,\alpha$.

Z6. The Zeckendorf representation is a *standard representation* of integers in the sense that if x is less than y, the representation for x has no more bits than the representation for y.

The following proposition gives an estimate of the extra space required if we use Zeckendorf representation instead of binary. Table 4 lists the space required by the Zeckendorf representation for a few values of r. We notice that the two representations become practically equivalent from $r = 6$ onward.

Proposition 1. For large enough n, in encoding a set of $F_{n+1}^{(r)}$ characters, the ratio of the space required by the Zeckendorf representation vs. the space required by the binary representation is $(\log_2(\lambda_1^{(r)}))^{-1}$ and tends to 1 as r increases.

r	2	3	4	5	6	7
	1.44	1.13	1.05	1.02	1.01	1.005

Table 4: Zeckendorf space / binary space.

0	00000	6	00110	12	01101	18	10101
1	00001	7	01000	13	10000	19	10110
2	00010	8	01001	14	10001	20	11000
3	00011	9	01010	15	10010	21	11001
4	00100	10	01011	16	10011	22	11010
5	00101	11	01100	17	10100	23	11011

Table 5: The Zeckendorf representation for the first $F_6^{(3)}$ numbers.

	r						
n	2	3	4	5	6	7	8
2	.5000	.5000	.5000	.5000	.5000	.5000	.5000
3	.3333	.5000	.5000	.5000	.5000	.5000	.5000
4	.3333	.4286	.5000	.5000	.5000	.5000	.5000
5	.3125	.4231	.4667	.5000	.5000	.5000	.5000
6	.3077	.4167	.4621	.4839	.5000	.5000	.5000
7	.3016	.4091	.4583	.4809	.4921	.5000	.5000
8	.2983	.4056	.4550	.4786	.4903	.4961	.5000
9	.2955	.4027	.4519	.4767	.4889	.4951	.4980
10	.2934	.4002	.4500	.4751	.4878	.4943	.4975
11	.2917	.3984	.4484	.4737	.4869	.4936	.4970
12	.2903	.3969	.4470	.4726	.4861	.4931	.4967
13	.2891	.3956	.4459	.4718	.4854	.4926	.4964
14	.2881	.3945	.4450	.4710	.4849	.4922	.4961
15	.2873	.3936	.4442	.4704	.4844	.4919	.4959
16	.2866	.3928	.4435	.4698	.4840	.4916	.4957
17	.2859	.3921	.4429	.4693	.4836	.4914	.4955
18	.2854	.3915	.4423	.4689	.4833	.4911	.4954
19	.2849	.3909	.4418	.4685	.4830	.4909	.4952
$A_\infty^{(r)}$	.2764	.3816	.4337	.4621	.4782	.4875	.4929

Table 6: Exact and asymptotic (for $n \to \infty$) proportion of 1's.

9 Asymptotic Proportion of ones

As we know, using the binary representation, the proportion of 0's in strings of length n and the proportion of 1's in strings of length n are both equal to $\frac{1}{2}$. If we use the Zeckendorf representation, these proportions are not equal. Table 5 illustrates the Zeckendorf representation of the first $F_6^{(3)} = 24$ natural numbers. By simple counting it can be seen that the preportion of $1's$ is .4167 (= 50/120). This corrosponds to $n = 6$ and $r = 3$ in Table 6.

The following theorem gives a complete characterization of the proportion of 1's in the Zeckendorf representation.

Theorem 1. The proportion of 1's in the Zeckendorf representation of integers is:

$$A_n^{(r)} = \frac{1}{\lambda_1^{(r)}} - \frac{r}{(\lambda_1^{(r)})^{r+1}} \frac{\lambda_1^{(r)} - 1}{[(r+1)\lambda_1^{(r)} - 2k]} + O\left(\frac{1}{n}\right) \tag{9}$$

and tends to $\frac{1}{2}$ as r increases.

Proof. Let $N(1)_n^{(r)}$ be the number of 1's and $N_n^{(r)}$ be the total number of sequences in R-nacci sequences of length $n+1$, we have

$$A_n^{(r)} = \frac{N(1)_n^{(r)}}{N_n^{(r)}}.$$

By induction on n it is possible to prove that the total number of 1's, $N(1)_n^{(r)}$, in Fibonacci sequences of length $n-1$ satisfies the following equation [2]:

$$\begin{aligned} N(1)_n^{(r)} &= N(1)_{n-1}^{(r)} + N(1)_{n-2}^{(r)} + F_{n-1}^{(r)} + N(1)_{n-3}^{(r)} + 2\,F_{n-2}^{(r)} + \ldots \\ &\quad + N(1)_{n-r+1}^{(r)} + (r-2)\,F_{n-r+2}^{(r)} + N(1)_{n-r}^{(r)} + (r-1)\,F_{n-r+1}^{(r)} \end{aligned}$$

which, applied recursively, gives:

$$N(1)_n^{(r)} = \sum_{j=0}^{n-2} (F_{j+1}^{(r)}) \sum_{i=1}^{r-1} i\,(F_{n-i-j}^{(r)}).$$

From fact Z1 in section 8 we know that the total number of symbols in R-nacci sequences of length $n-1$ is $(n-1)F_{n+1}^{(r)}$. Thus we have:

$$A_n^{(r)} = \frac{\sum_{j=0}^{n-2} (F_{j+1}^{(r)}) \sum_{i=1}^{r-1} i\,(F_{n-i-j}^{(r)})}{(n-1)F_{n+1}^{(r)}} \tag{10}$$

Using approximation (8) and after some algebra, (10) becomes:

$$A_n^{(r)} = b_1^{(r)} \sum_{i=1}^{r-1} i\,(\lambda_i^{(r)})^{-i} + O(\frac{1}{n}) \tag{11}$$

where $b_1 = \frac{\lambda^{(r)}-1}{\lambda^{(r)}[(r+1)\lambda^{(r)}-2r]}$. Since

$$\sum_{i=1}^{r-1} i\,(\lambda_i^{(r)})^{-i} = (b_1^{(r)}\,\lambda_1^{(r)})^{-1} - r\,(\lambda_1^{(r)})^{-r},$$

(11) becomes:

$$A_n^{(r)} = (\lambda_1^{(r)})^{-1} - r\,b_1^{(r)}\,(\lambda_1^{(r)})^{-r} + O(\frac{1}{n}) \tag{12}$$

and using fact R1 in section 7 we get $\lim_{r,n\to\infty} A_n^{(r)} = \frac{1}{2}$. Finally, it is possible to prove that $(\lambda_i^{(r)})^r = 1/(2 - (\lambda_i^{(r)}))$ which substituted into (12) proves the theorem. □

The asymptotic proportion of 1's in R-nacci sequences has been recently calculated by Chang [4] using a somewhat different method. He found a considerably more complex algebraic expression. Moreover, our formula seems to give more accurate numerical values. Indeed, numerical values obtained from Chang's formula do not converge increasingly to 1/2 as is to be expected. This is essentially due to the fact that Chang uses the identity $(2-\lambda_2^{(r)})\ldots(2-\lambda_r^{(r)}) = 1/(2-\lambda_1^{(r)})$, and we use the identity $(\lambda_1^{(r)})^r = 1/(2-\lambda_1^{(r)})$. The identity used by Chang is more sensitive to rounding errors. Table 6 lists for $1 \le r \le 8$ the values of the exact proportion of 1's in a code for $F_n^{(r)}$ characters and the asymptotic proportion of 1's for $n \to \infty$. It must be noted that for $n \le r$ the proportion is $\frac{1}{2}$.

Numerically $A_n^{(r)}$ is monotone decreasing in n. We can show that monotonicity must occur asymptotically, but we cannot show that $A_n^{(r)}$ always decreases. For the usual Fibonacci numbers

$$A_n^{(2)} = 1 - \frac{nl_{n+3} - 2f_n}{5nf_{n+2}}$$

where l_{n+3} is a Lucas number. From this fromula we can *prove* that $A_n^{(2)}$ is monotone decreasing, but we use $\lambda_1\lambda_2 = -1$ and for $k > 2$, $|\lambda_1\lambda_2| > 1$ is possible.

References

[1] Aho, A. V., J. E. Hopcroft and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading MA; 1974.

[2] Capocelli, R. M., "A generalization of Fibonacci Trees," Third International Conference on Fibonacci Numbers and Their Application, Pisa, Italy; 1988.

[3] Capocelli, R. M., and P. Cull, "Generalized Fibonacci numbers are rounded powers," Third International Conference on Fibonacci Numbers and Their Applications, Pisa, Italy; 1988.

[4] Chang, D. K., "On Fibonacci Binary Sequences," *The Fibonacci Quarterly* 24, 178-179; 1986.

[5] Er, M. C., "A fast algorithm for computing order-k Fibonacci numbers," *The Computer Journal* **26**, 224-227; 1983.

[6] Feinberg, M., "Fibonacci-Tribonacci," *The Fibonacci Quarterly* **1**, 71-74; 1963.

[7] Gries, D. and G. Levin, "Computing Fibonacci numbers (and similarly defined functions) in log time," *Information Processing Letters* **11**, 68-69; 1980.

[8] Hoggat Jr., V. E., and M. Bicknell, "Generalized Fibonacci Polynomials and Zeckendorf's Representations," *The Fibonacci Quarterly* **11**, 399-419; 1973.

[9] Knuth, D. E., "The Art of Computer Programming," Vols. 1 and 2, Addison-Wesley, Reading MA; 1973, 1981.

[10] Pettorossi, A., "Derivation of an $O(k^2 \log n)$ algorithm for computing order-k Fibonacci numbers from the $O(k^3 \log n)$ matrix multiplication method," *Information Processing Letters* **11**, 172-179; 1980.

[11] Romano, A., "Applied statistics for science and industry," Allyn and Bacon, Boston; 1977.

[12] Shortt J., "An iterative program to calculate Fibonacci Numbers in $O(\log n)$ arithmetic operations," *Information Processing Letters* **7**, 299-303; 1977.

[13] Spickerman, W. R., "Binet's Formula for the Tribonacci Sequence," *The Fibonacci Quarterly* **20**, 118-120; 1982.

[14] Urbanek, F. J., "An $O(\log n)$ algorithm for computing the n^{th} element of the solution of a difference equation," *Information Processing Letters* **11**, 66-67; 1980.

[15] Vorob'ev, N.N., "Fibonacci Numbers," Pergamon Press, New York; 1961.

[16] Welch, T. A., "A technique for high-perfomance data compression," *Computer* **17**, 8-19; 1984.

[17] Zeckendorf, E., "Representations des Nombres Naturels par une Somme de Nombres de Fibonacci ou de Nombres de Lucas," *Bulletin de la Societé Royale des Sciences de Liege* 179-182; 1972.

[18] Ziv, J. and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory* **IT-24**, 530-536; 1978.