Representing Numbers Using Fibonacci Variants

Stephen Lucas

Department of Mathematics and Statistics
James Madison University, Harrisonburg VA

September 9 2013

Outline

- Fibonacci Numbers
- Zeckendorf Form and Fibonacci Coding
- Continued Fractions
- Generalizing Fibonacci Coding
- Arithmetic

Fibonacci Numbers

Fibonacci numbers satisfy $f_n = f_{n-1} + f_{n-2}$ with $f_0 = 0$, $f_1 = 1$.

Fibonacci Numbers

Fibonacci numbers satisfy $f_n = f_{n-1} + f_{n-2}$ with $f_0 = 0$, $f_1 = 1$.

In closed form, $f_k = \dfrac{\phi^k - (1-\phi)^k}{\sqrt{5}}$ where $\phi = \dfrac{1+\sqrt{5}}{2}$ is the golden ratio.

Fibonacci Numbers

Fibonacci numbers satisfy $f_n = f_{n-1} + f_{n-2}$ with $f_0 = 0$, $f_1 = 1$.

In closed form, $f_k = \dfrac{\phi^k - (1-\phi)^k}{\sqrt{5}}$ where $\phi = \dfrac{1+\sqrt{5}}{2}$ is the golden ratio.

The sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Discovered by Eduourd Zeckendorf in 1939, published by him in 1972, first published (in German) in 1952.

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Discovered by Eduourd Zeckendorf in 1939, published by him in 1972, first published (in German) in 1952.

Greedy algorithm: choose largest Fibonacci number less than remaining, subtract, repeat.

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Discovered by Eduourd Zeckendorf in 1939, published by him in 1972, first published (in German) in 1952.

Greedy algorithm: choose largest Fibonacci number less than remaining, subtract, repeat.

For example, 825.

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Discovered by Eduourd Zeckendorf in 1939, published by him in 1972, first published (in German) in 1952.

Greedy algorithm: choose largest Fibonacci number less than remaining, subtract, repeat.

For example, 825. $f_{15} = 610$, $825 - 610 = 215$.

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Discovered by Eduourd Zeckendorf in 1939, published by him in 1972, first published (in German) in 1952.

Greedy algorithm: choose largest Fibonacci number less than remaining, subtract, repeat.

For example, 825. $f_{15} = 610$, $825 - 610 = 215$. $f_{12} = 144$, $215 - 144 = 71$.

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Discovered by Eduourd Zeckendorf in 1939, published by him in 1972, first published (in German) in 1952.

Greedy algorithm: choose largest Fibonacci number less than remaining, subtract, repeat.

For example, 825. $f_{15} = 610$, $825 - 610 = 215$. $f_{12} = 144$, $215 - 144 = 71$. $f_{10} = 55$, $71 - 55 = 16$.

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Discovered by Eduourd Zeckendorf in 1939, published by him in 1972, first published (in German) in 1952.

Greedy algorithm: choose largest Fibonacci number less than remaining, subtract, repeat.

For example, 825. $f_{15} = 610$, $825 - 610 = 215$. $f_{12} = 144$, $215 - 144 = 71$. $f_{10} = 55$, $71 - 55 = 16$. $f_7 = 13$, $16 - 13 = 3$.

Zeckendorf Form

Every natural number can be uniquely represented by a sum of distinct non-consecutive Fibonacci numbers, starting from $f_2 = 1$.

Discovered by Eduourd Zeckendorf in 1939, published by him in 1972, first published (in German) in 1952.

Greedy algorithm: choose largest Fibonacci number less than remaining, subtract, repeat.

For example, 825. $f_{15} = 610$, $825 - 610 = 215$. $f_{12} = 144$, $215 - 144 = 71$. $f_{10} = 55$, $71 - 55 = 16$. $f_7 = 13$, $16 - 13 = 3$. $f_4 = 3$, so $825 = f_{15} + f_{12} + f_{10} + f_7 + f_4$, or $(10010100100100)_Z$.

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.
Assume every integer from 1 to n has a Zeckendorf representation.

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.
Assume every integer from 1 to n has a Zeckendorf representation.
If $n + 1$ is a Fibonacci number, done.

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.
Assume every integer from 1 to n has a Zeckendorf representation.
If $n + 1$ is a Fibonacci number, done. Otherwise, there is some j such that $f_j < n + 1 < f_{j+1}$.

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.
Assume every integer from 1 to n has a Zeckendorf representation. If $n + 1$ is a Fibonacci number, done. Otherwise, there is some j such that $f_j < n + 1 < f_{j+1}$. Now $n + 1 - f_j < n$, so has a Zeckendorf representation,

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.
Assume every integer from 1 to n has a Zeckendorf representation.
If $n+1$ is a Fibonacci number, done. Otherwise, there is some j such that $f_j < n+1 < f_{j+1}$. Now $n+1-f_j < n$, so has a Zeckendorf representation, and $n+1-f_j < f_{j+1}-f_j = f_{j-1}$, so $n+1-f_j$ doesn't contain f_j, done.

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.
Assume every integer from 1 to n has a Zeckendorf representation. If $n+1$ is a Fibonacci number, done. Otherwise, there is some j such that $f_j < n+1 < f_{j+1}$. Now $n+1-f_j < n$, so has a Zeckendorf representation, and $n+1-f_j < f_{j+1} - f_j = f_{j-1}$, so $n+1-f_j$ doesn't contain f_j, done.

Uniqueness: We need that the sum of distinct non-consecutive Fibonacci numbers up to f_n is less than f_{n+1} (induction).

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.
Assume every integer from 1 to n has a Zeckendorf representation. If $n + 1$ is a Fibonacci number, done. Otherwise, there is some j such that $f_j < n + 1 < f_{j+1}$. Now $n + 1 - f_j < n$, so has a Zeckendorf representation, and $n + 1 - f_j < f_{j+1} - f_j = f_{j-1}$, so $n + 1 - f_j$ doesn't contain f_j, done.

Uniqueness: We need that the sum of distinct non-consecutive Fibonacci numbers up to f_n is less than f_{n+1} (induction). Assume two different sets with the same sum, eliminate common numbers.

Proofs

Existence by induction: $1 = f_2$, $2 = f_3$, $3 = f_4$.
Assume every integer from 1 to n has a Zeckendorf representation. If $n+1$ is a Fibonacci number, done. Otherwise, there is some j such that $f_j < n+1 < f_{j+1}$. Now $n+1-f_j < n$, so has a Zeckendorf representation, and $n+1-f_j < f_{j+1} - f_j = f_{j-1}$, so $n+1-f_j$ doesn't contain f_j, done.

Uniqueness: We need that the sum of distinct non-consecutive Fibonacci numbers up to f_n is less than f_{n+1} (induction). Assume two different sets with the same sum, eliminate common numbers. The largest (in one set) must be larger than the collection in the other set, so the two sums cannot be the same!

Efficiency

Zeckendorf representation of a number is a string of zeros and (non-consecutive) ones.

Efficiency

Zeckendorf representation of a number is a string of zeros and (non-consecutive) ones. It doesn't formally have a base. But, since f_k is the closest natural number to $\phi^k/\sqrt{5}$, the ratio of Fibonacci numbers approaches ϕ.

Efficiency

Zeckendorf representation of a number is a string of zeros and (non-consecutive) ones. It doesn't formally have a base. But, since f_k is the closest natural number to $\phi^k/\sqrt{5}$, the ratio of Fibonacci numbers approaches ϕ. Thus Zeckendorf representation has roughly base $\phi \approx 1.618 < 2$, less efficient than binary.

Efficiency

Zeckendorf representation of a number is a string of zeros and (non-consecutive) ones. It doesn't formally have a base. But, since f_k is the closest natural number to $\phi^k/\sqrt{5}$, the ratio of Fibonacci numbers approaches ϕ. Thus Zeckendorf representation has roughly base $\phi \approx 1.618 < 2$, less efficient than binary.

But Zeckendorf representation can't contain a pair of consecutive ones, so a pair can be used to separate numbers in a list, using a variable number of digits per number.

Efficiency

Zeckendorf representation of a number is a string of zeros and (non-consecutive) ones. It doesn't formally have a base. But, since f_k is the closest natural number to $\phi^k/\sqrt{5}$, the ratio of Fibonacci numbers approaches ϕ. Thus Zeckendorf representation has roughly base $\phi \approx 1.618 < 2$, less efficient than binary.

But Zeckendorf representation can't contain a pair of consecutive ones, so a pair can be used to separate numbers in a list, using a variable number of digits per number. Fibonacci coding reverses the order of digits, so is always a trailing one, and only one extra one is "wasted" separating numbers.

Efficiency

Zeckendorf representation of a number is a string of zeros and (non-consecutive) ones. It doesn't formally have a base. But, since f_k is the closest natural number to $\phi^k/\sqrt{5}$, the ratio of Fibonacci numbers approaches ϕ. Thus Zeckendorf representation has roughly base $\phi \approx 1.618 < 2$, less efficient than binary.

But Zeckendorf representation can't contain a pair of consecutive ones, so a pair can be used to separate numbers in a list, using a variable number of digits per number. Fibonacci coding reverses the order of digits, so is always a trailing one, and only one extra one is "wasted" separating numbers.

E.g. 10010101110001011011

Efficiency

Zeckendorf representation of a number is a string of zeros and (non-consecutive) ones. It doesn't formally have a base. But, since f_k is the closest natural number to $\phi^k/\sqrt{5}$, the ratio of Fibonacci numbers approaches ϕ. Thus Zeckendorf representation has roughly base $\phi \approx 1.618 < 2$, less efficient than binary.

But Zeckendorf representation can't contain a pair of consecutive ones, so a pair can be used to separate numbers in a list, using a variable number of digits per number. Fibonacci coding reverses the order of digits, so is always a trailing one, and only one extra one is "wasted" separating numbers.

E.g. 1001010111000101 1011 represents 10010101, 1000101, and 01, or $f_2 + f_5 + f_7 + f_9$, $f_2 + f_6 + f_8$, f_3, or 53, 30, 2.

Representing Numbers from Distributions

Representing Numbers from Distributions

- Fibonacci coding is particularly useful when there is no prior knowledge of the upper bound on numbers from a list.

Representing Numbers from Distributions

- Fibonacci coding is particularly useful when there is no prior knowledge of the upper bound on numbers from a list.
- Numbers uniformly distributed from one to a million: Fibonacci coding 27.8 bits per number, binary 20.

Representing Numbers from Distributions

- Fibonacci coding is particularly useful when there is no prior knowledge of the upper bound on numbers from a list.
- Numbers uniformly distributed from one to a million: Fibonacci coding 27.8 bits per number, binary 20.
- One to ten equally likely, 10^6 one in ten thousand: Fibonacci coding 4.6 bits per number, binary still 20.

Representing Numbers from Distributions

- Fibonacci coding is particularly useful when there is no prior knowledge of the upper bound on numbers from a list.
- Numbers uniformly distributed from one to a million: Fibonacci coding 27.8 bits per number, binary 20.
- One to ten equally likely, 10^6 one in ten thousand: Fibonacci coding 4.6 bits per number, binary still 20.
- Numbers Poisson with $\lambda = 4$: $\Pr(X = k) = \lambda^k e^{-\lambda}/k!$. Numerically 0 to 31:

Representing Numbers from Distributions

- Fibonacci coding is particularly useful when there is no prior knowledge of the upper bound on numbers from a list.
- Numbers uniformly distributed from one to a million: Fibonacci coding 27.8 bits per number, binary 20.
- One to ten equally likely, 10^6 one in ten thousand: Fibonacci coding 4.6 bits per number, binary still 20.
- Numbers Poisson with $\lambda = 4$: $\Pr(X = k) = \lambda^k e^{-\lambda}/k!$. Numerically 0 to 31: Fibonacci coding 4.6 bits per number, binary 5.

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$.

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$. $236 = 9 \times 24 + 20$,

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$. $236 = 9 \times 24 + 20$, $24 = 1 \times 20 + 4$,

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$. $236 = 9 \times 24 + 20$, $24 = 1 \times 20 + 4$, $20 = 5 \times 4 + 0$,

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$. $236 = 9 \times 24 + 20$, $24 = 1 \times 20 + 4$, $20 = 5 \times 4 + 0$, so $\gcd(236, 24) = 4$.

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$. $236 = 9 \times 24 + 20$, $24 = 1 \times 20 + 4$, $20 = 5 \times 4 + 0$, so $\gcd(236, 24) = 4$.

Or

$$\frac{236}{24} = 9 + \frac{20}{24}$$

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$. $236 = 9 \times 24 + 20$, $24 = 1 \times 20 + 4$, $20 = 5 \times 4 + 0$, so $\gcd(236, 24) = 4$.

Or

$$\frac{236}{24} = 9 + \frac{20}{24} = 9 + \frac{1}{24/20} = 9 + \frac{1}{1 + \frac{4}{20}}$$

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$. $236 = 9 \times 24 + 20$, $24 = 1 \times 20 + 4$, $20 = 5 \times 4 + 0$, so $\gcd(236, 24) = 4$.

Or

$$\begin{aligned}\frac{236}{24} &= 9 + \frac{20}{24} = 9 + \frac{1}{24/20} = 9 + \frac{1}{1 + \frac{4}{20}} \\ &= 9 + \frac{1}{1 + \frac{1}{20/4}} = 9 + \frac{1}{1 + \frac{1}{5}}\end{aligned}$$

Greatest Common Divisor

Continued Fractions are closely related to the greatest common divisor algorithm.

For example, consider $\gcd(236, 24)$. $236 = 9 \times 24 + 20$, $24 = 1 \times 20 + 4$, $20 = 5 \times 4 + 0$, so $\gcd(236, 24) = 4$.

Or

$$\begin{aligned}\frac{236}{24} &= 9 + \frac{20}{24} = 9 + \frac{1}{24/20} = 9 + \frac{1}{1 + \frac{4}{20}} \\ &= 9 + \frac{1}{1 + \frac{1}{20/4}} = 9 + \frac{1}{1 + \frac{1}{5}} = [9; 1, 5].\end{aligned}$$

Continued Fractions

A simple continued fraction for a (positive) fraction is

$$\frac{p}{q} = b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{\vdots}{b_{n-1} + \cfrac{1}{b_n}}}} \equiv b_0 + \frac{1}{b_1 +} \frac{1}{b_2 +} \cdots + \frac{1}{b_n} \equiv [b_0; b_1, b_2, \ldots, b_n],$$

where b_0 is an integer, and the b_i's for $i > 0$ are natural numbers. The b_i's are traditionally called partial quotients.

Continued Fractions

A simple continued fraction for a (positive) fraction is

$$\frac{p}{q} = b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{\vdots}{b_{n-1} + \cfrac{1}{b_n}}}} \equiv b_0 + \frac{1}{b_1 +}\frac{1}{b_2 +} \cdots + \frac{1}{b_n} \equiv [b_0; b_1, b_2, \ldots, b_n],$$

where b_0 is an integer, and the b_i's for $i > 0$ are natural numbers. The b_i's are traditionally called partial quotients.

Algorithm: given x, set $x_0 = x$ and $b_0 = \lfloor x_0 \rfloor$, then

$$x_i = \frac{1}{x_{i-1} - b_{i-1}} \quad \text{and} \quad b_i = \lfloor x_i \rfloor \quad \text{for} \quad i = 1, 2, \ldots,$$

until some x_i is an integer.

Arbitrary Irrationals

The continued fraction of a rational is finite, but the algorithm can also be applied to irrationals, and forms an infinite sequence.

Arbitrary Irrationals

The continued fraction of a rational is finite, but the algorithm can also be applied to irrationals, and forms an infinite sequence.

Continued fractions have many elegant features, including the Gauss-Kuzmin theorem: for almost all irrationals between zero and one,

$$\lim_{n\to\infty} P(k_n = k) = -\log_2\left(1 - \frac{1}{(k+1)^2}\right).$$

Arbitrary Irrationals

The continued fraction of a rational is finite, but the algorithm can also be applied to irrationals, and forms an infinite sequence.

Continued fractions have many elegant features, including the Gauss-Kuzmin theorem: for almost all irrationals between zero and one,

$$\lim_{n\to\infty} P(k_n = k) = -\log_2\left(1 - \frac{1}{(k+1)^2}\right).$$

Arbitrarily large partial quotients are possible, but increasingly unlikely.

Arbitrary Irrationals

The continued fraction of a rational is finite, but the algorithm can also be applied to irrationals, and forms an infinite sequence.

Continued fractions have many elegant features, including the Gauss-Kuzmin theorem: for almost all irrationals between zero and one,

$$\lim_{n\to\infty} P(k_n = k) = -\log_2\left(1 - \frac{1}{(k+1)^2}\right).$$

Arbitrarily large partial quotients are possible, but increasingly unlikely. Fibonacci coding is an ideal choice for representing continued fraction partial quotients for arbitrary irrationals.

Gauss-Kuzmin Distribution

k	Prob.	k	Prob.
1	0.415037	10	0.011973
2	0.169925	100	1.41434×10^{-4}
3	0.093109	1000	1.43981×10^{-6}
4	0.058894	10 000	1.44241×10^{-8}
5	0.040642		
6	0.029747	> 10	1.25531×10^{-1}
7	0.022720	> 100	1.42139×10^{-2}
8	0.017922	> 1000	1.44053×10^{-3}
9	0.014500	$> 10\,000$	1.44248×10^{-4}

Examples

- The first 20 000 partial quotients of ln(2) has largest partial quotient 963 664.

Examples

- The first 20 000 partial quotients of ln(2) has largest partial quotient 963 664. In binary, 20 bits per partial quotient (with previous knowledge).

Examples

- The first 20 000 partial quotients of ln(2) has largest partial quotient 963 664. In binary, 20 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.74 bits.

Examples

- The first 20 000 partial quotients of ln(2) has largest partial quotient 963 664. In binary, 20 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.74 bits.
- The first 20 000 partial quotients of π has largest partial quotient 74 174. In binary, 17 bits per partial quotient (with previous knowledge).

Examples

- The first 20 000 partial quotients of ln(2) has largest partial quotient 963 664. In binary, 20 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.74 bits.
- The first 20 000 partial quotients of π has largest partial quotient 74 174. In binary, 17 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.71 bits.

Examples

- The first 20 000 partial quotients of ln(2) has largest partial quotient 963 664. In binary, 20 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.74 bits.
- The first 20 000 partial quotients of π has largest partial quotient 74 174. In binary, 17 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.71 bits.

Loch's theorem: $\displaystyle\lim_{n\to\infty} \frac{\#\ \text{partial quotients}}{\#\ \text{correct binary digits}} = \frac{6(\ln 2)^2}{\pi^2} \approx 0.292.$

Examples

- The first 20 000 partial quotients of ln(2) has largest partial quotient 963 664. In binary, 20 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.74 bits.
- The first 20 000 partial quotients of π has largest partial quotient 74 174. In binary, 17 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.71 bits.

Loch's theorem: $\lim_{n\to\infty} \frac{\#\text{ partial quotients}}{\#\text{ correct binary digits}} = \frac{6(\ln 2)^2}{\pi^2} \approx 0.292.$
Or, about 3.42 times number of partial quotients bits in binary,

Examples

- The first 20 000 partial quotients of ln(2) has largest partial quotient 963 664. In binary, 20 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.74 bits.
- The first 20 000 partial quotients of π has largest partial quotient 74 174. In binary, 17 bits per partial quotient (with previous knowledge). Fibonacci coding requires 3.71 bits.

Loch's theorem: $\lim_{n\to\infty} \frac{\#\text{ partial quotients}}{\#\text{ correct binary digits}} = \frac{6(\ln 2)^2}{\pi^2} \approx 0.292.$
Or, about 3.42 times number of partial quotients bits in binary, Slightly more efficient (if you don't want the continued fraction data).

Generalized Fibonacci Coding

The effective base of Zeckendorf form is $\phi \approx 1.618$.

Generalized Fibonacci Coding

The effective base of Zeckendorf form is $\phi \approx 1.618$.

Tribonacci numbers satisfy $t_n = t_{n-1} + t_{n-2} + t_{n-3}$ with $t_{-1} = t_0 = 0,\ t_1 = 1$, and grow like 1.8393^n.

Generalized Fibonacci Coding

The effective base of Zeckendorf form is $\phi \approx 1.618$.

Tribonacci numbers satisfy $t_n = t_{n-1} + t_{n-2} + t_{n-3}$ with $t_{-1} = t_0 = 0$, $t_1 = 1$, and grow like 1.8393^n.
Tetranacci numbers satisfy $u_n = u_{n-1} + u_{n-2} + u_{n-3} + u_{n-4}$ with $u_{-2} = u_{-1} = u_0 = 0$, $u_1 = 1$, and grow like 1.9276^n.

Generalized Fibonacci Coding

The effective base of Zeckendorf form is $\phi \approx 1.618$.

Tribonacci numbers satisfy $t_n = t_{n-1} + t_{n-2} + t_{n-3}$ with $t_{-1} = t_0 = 0,\ t_1 = 1$, and grow like 1.8393^n.
Tetranacci numbers satisfy $u_n = u_{n-1} + u_{n-2} + u_{n-3} + u_{n-4}$ with $u_{-2} = u_{-1} = u_0 = 0,\ u_1 = 1$, and grow like 1.9276^n.

k-bonacci numbers satisfy $u_n = \sum_{i=1}^{n} u_{n-i},\ u_1 = 1,\ u_i = 0$ for $i < 0$.

Generalized Fibonacci Coding

The effective base of Zeckendorf form is $\phi \approx 1.618$.

Tribonacci numbers satisfy $t_n = t_{n-1} + t_{n-2} + t_{n-3}$ with $t_{-1} = t_0 = 0,\ t_1 = 1$, and grow like 1.8393^n.
Tetranacci numbers satisfy $u_n = u_{n-1} + u_{n-2} + u_{n-3} + u_{n-4}$ with $u_{-2} = u_{-1} = u_0 = 0,\ u_1 = 1$, and grow like 1.9276^n.

k-bonacci numbers satisfy $u_n = \sum_{i=1}^{n} u_{n-i},\ u_1 = 1,\ u_i = 0$ for $i < 0$.

Numbers can be uniquely represented by sums of k-bonacci numbers with no k ones in a row. So, k-bonacci coding uses $k-1$ digits to separate numbers in variable length encoding.

Examples

- Uniformly distributed one to a million (binary 20)

k	2	3	4	5
Bits	27.82	23.34	22.86	23.40

Examples

- Uniformly distributed one to a million (binary 20)

k	2	3	4	5
Bits	27.82	23.34	22.86	23.40

- One to ten equally likely, 10^6 one in ten thousand (binary 20)

k	2	3	4	5
Bits	4.60	5.00	5.90	6.90

Examples

- Uniformly distributed one to a million (binary 20)

k	2	3	4	5
Bits	27.82	23.34	22.86	23.40

- One to ten equally likely, 10^6 one in ten thousand (binary 20)

k	2	3	4	5
Bits	4.60	5.00	5.90	6.90

- Poisson $\lambda = 4$ (binary 5)

k	2	3	4	5
Bits	4.57	4.96	5.85	6.85

Examples

- Uniformly distributed one to a million (binary 20)

k	2	3	4	5
Bits	27.82	23.34	22.86	23.40

- One to ten equally likely, 10^6 one in ten thousand (binary 20)

k	2	3	4	5
Bits	4.60	5.00	5.90	6.90

- Poisson $\lambda = 4$ (binary 5)

k	2	3	4	5
Bits	4.57	4.96	5.85	6.85

- ln(2) partial quotients (binary 20 or 3.42)

k	2	3	4	5
Bits	3.74	4.35	5.28	6.26

Examples

- Uniformly distributed one to a million (binary 20)

k	2	3	4	5
Bits	27.82	23.34	22.86	23.40

- One to ten equally likely, 10^6 one in ten thousand (binary 20)

k	2	3	4	5
Bits	4.60	5.00	5.90	6.90

- Poisson $\lambda = 4$ (binary 5)

k	2	3	4	5
Bits	4.57	4.96	5.85	6.85

- ln(2) partial quotients (binary 20 or 3.42)

k	2	3	4	5
Bits	3.74	4.35	5.28	6.26

- π partial quotients (binary 17 or 3.42)

k	2	3	4	5
Bits	3.71	4.33	5.27	6.25

Examples

- Uniformly distributed one to a million (binary 20)

k	2	3	4	5
Bits	27.82	23.34	22.86	23.40

- One to ten equally likely, 10^6 one in ten thousand (binary 20)

k	2	3	4	5
Bits	4.60	5.00	5.90	6.90

- Poisson $\lambda = 4$ (binary 5)

k	2	3	4	5
Bits	4.57	4.96	5.85	6.85

- ln(2) partial quotients (binary 20 or 3.42)

k	2	3	4	5
Bits	3.74	4.35	5.28	6.26

- π partial quotients (binary 17 or 3.42)

k	2	3	4	5
Bits	3.71	4.33	5.27	6.25

Not really worth it ☹

JAMES MADISON UNIVERSITY

Arithmetic

Arithmetic is possible on numbers in Zeckendorf form, and is most easily done on a checkerboard. Manipulating digits is easy, but numbers need to be returned to Zeckendorf form.

Arithmetic

Arithmetic is possible on numbers in Zeckendorf form, and is most easily done on a checkerboard. Manipulating digits is easy, but numbers need to be returned to Zeckendorf form.

- Pair rule: Since $f_n - f_{n-1} - f_{n-2} = 0$, subtracting one from successive digits adds one to the one to the left, or $(\ldots(+1)(-1)(-1)\ldots)_Z$.

Arithmetic

Arithmetic is possible on numbers in Zeckendorf form, and is most easily done on a checkerboard. Manipulating digits is easy, but numbers need to be returned to Zeckendorf form.

- Pair rule: Since $f_n - f_{n-1} - f_{n-2} = 0$, subtracting one from successive digits adds one to the one to the left, or $(\ldots(+1)(-1)(-1)\ldots)_Z$.
- Two rule: Subtracting $f_{n+1} = f_n + f_{n-1}$ from $f_n = f_{n-1} + f_{n-2}$, $f_{n+1} + f_{n-2} - 2f_n = 0$, or $(\ldots(+1)(-2)(0)(+1)\ldots)_Z$.

Arithmetic

Arithmetic is possible on numbers in Zeckendorf form, and is most easily done on a checkerboard. Manipulating digits is easy, but numbers need to be returned to Zeckendorf form.

- Pair rule: Since $f_n - f_{n-1} - f_{n-2} = 0$, subtracting one from successive digits adds one to the one to the left, or $(\ldots(+1)(-1)(-1)\ldots)_Z$.
- Two rule: Subtracting $f_{n+1} = f_n + f_{n-1}$ from $f_n = f_{n-1} + f_{n-2}$, $f_{n+1} + f_{n-2} - 2f_n = 0$, or $(\ldots(+1)(-2)(0)(+1)\ldots)_Z$.
- Edge two rule: $(\ldots(+1)(-2))_Z$ and $(\ldots(+1)(-2)(+1))_Z$.

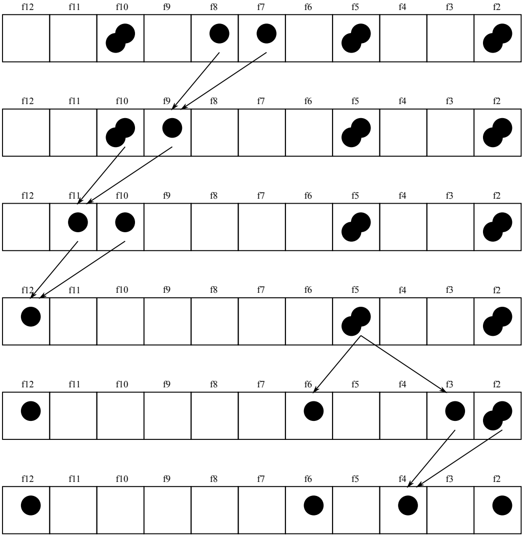
Arithmetic

Arithmetic is possible on numbers in Zeckendorf form, and is most easily done on a checkerboard. Manipulating digits is easy, but numbers need to be returned to Zeckendorf form.

- Pair rule: Since $f_n - f_{n-1} - f_{n-2} = 0$, subtracting one from successive digits adds one to the one to the left, or $(\ldots(+1)(-1)(-1)\ldots)_Z$.
- Two rule: Subtracting $f_{n+1} = f_n + f_{n-1}$ from $f_n = f_{n-1} + f_{n-2}$, $f_{n+1} + f_{n-2} - 2f_n = 0$, or $(\ldots(+1)(-2)(0)(+1)\ldots)_Z$.
- Edge two rule: $(\ldots(+1)(-2))_Z$ and $(\ldots(+1)(-2)(+1))_Z$.

For example, $(101001001)_F + (100101001)_F = (201102002)_F$.

Addition Example



Order of Applying Rules

We didn't use a systematic approach to simplifying addition.

Order of Applying Rules

We didn't use a systematic approach to simplifying addition.

- Tee (2002): right to left with recursive two rule, $O(n^3)$.

Order of Applying Rules

We didn't use a systematic approach to simplifying addition.

- Tee (2002): right to left with recursive two rule, $O(n^3)$.
- Ahlbach *et al.* (2012 arxiv): three passes. First left to right, $020x \to 100(x+1)$, $030x \to 110(x+1)$, $021x \to 110x$, $012x \to 101x$, eliminates twos.

Order of Applying Rules

We didn't use a systematic approach to simplifying addition.

- Tee (2002): right to left with recursive two rule, $O(n^3)$.
- Ahlbach *et al.* (2012 arxiv): three passes. First left to right, $020x \to 100(x+1)$, $030x \to 110(x+1)$, $021x \to 110x$, $012x \to 101x$, eliminates twos. Second right to left, third left to right, $011 \to 100$.

Order of Applying Rules

We didn't use a systematic approach to simplifying addition.

- Tee (2002): right to left with recursive two rule, $O(n^3)$.
- Ahlbach *et al.* (2012 arxiv): three passes. First left to right, $020x \to 100(x+1)$, $030x \to 110(x+1)$, $021x \to 110x$, $012x \to 101x$, eliminates twos. Second right to left, third left to right, $011 \to 100$. Second pass eliminates 1011 pattern.

Order of Applying Rules

We didn't use a systematic approach to simplifying addition.

- Tee (2002): right to left with recursive two rule, $O(n^3)$.
- Ahlbach *et al.* (2012 arxiv): three passes. First left to right, $020x \to 100(x+1)$, $030x \to 110(x+1)$, $021x \to 110x$, $012x \to 101x$, eliminates twos. Second right to left, third left to right, $011 \to 100$. Second pass eliminates 1011 pattern.
- Lucas (now): two passes. First as Ahlbach *et al.*.

Order of Applying Rules

We didn't use a systematic approach to simplifying addition.

- Tee (2002): right to left with recursive two rule, $O(n^3)$.
- Ahlbach *et al.* (2012 arxiv): three passes. First left to right, $020x \to 100(x+1)$, $030x \to 110(x+1)$, $021x \to 110x$, $012x \to 101x$, eliminates twos. Second right to left, third left to right, $011 \to 100$. Second pass eliminates 1011 pattern.
- Lucas (now): two passes. First as Ahlbach *et al.*. Second insert leading 0, then left to right, $(01)^k1 \to 1(0)^{2k}$.

Order of Applying Rules

We didn't use a systematic approach to simplifying addition.

- Tee (2002): right to left with recursive two rule, $O(n^3)$.
- Ahlbach *et al.* (2012 arxiv): three passes. First left to right, $020x \rightarrow 100(x+1)$, $030x \rightarrow 110(x+1)$, $021x \rightarrow 110x$, $012x \rightarrow 101x$, eliminates twos. Second right to left, third left to right, $011 \rightarrow 100$. Second pass eliminates 1011 pattern.
- Lucas (now): two passes. First as Ahlbach *et al.*. Second insert leading 0, then left to right, $(01)^k1 \rightarrow 1(0)^{2k}$. A pair of zeros means move pointer to right.

Subtraction

In columns, $0 - 0 = 0$, $1 - 0 = 1$, $1 - 1 = 0$.

Subtraction

In columns, $0 - 0 = 0$, $1 - 0 = 1$, $1 - 1 = 0$.

If $0 - 1$, use reallocation as with standard subtraction (at most three passes), then $1 - 1 = 0$.

Subtraction

In columns, $0-0=0$, $1-0=1$, $1-1=0$.

If $0-1$, use reallocation as with standard subtraction (at most three passes), then $1-1=0$.

All ones, finally one Lucas pass, back to Zeckendorf form.

Subtraction

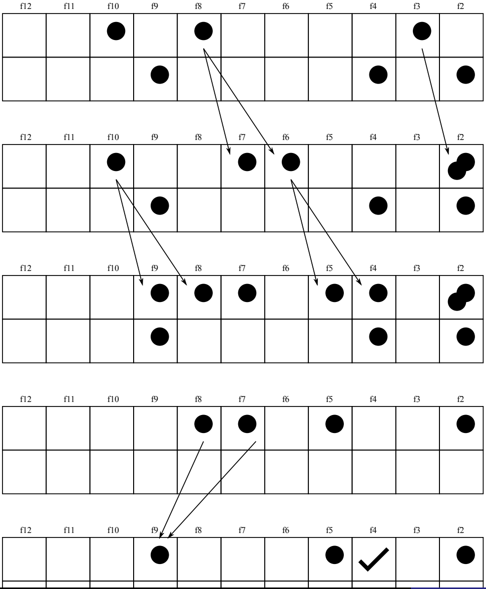
In columns, $0 - 0 = 0$, $1 - 0 = 1$, $1 - 1 = 0$.

If $0 - 1$, use reallocation as with standard subtraction (at most three passes), then $1 - 1 = 0$.

All ones, finally one Lucas pass, back to Zeckendorf form.

Fenwick (2003) introduces a difficult complement, Ahlbach *et al.* just subtract digits, add another pass to eliminate negative digits. Tee also thought it was $O(n^3)$.

Subtraction Example



Multiplication Four Ways

- Freitag and Phillips (1998): Multiplication digit by digit, using

$$m \geq 2i : \qquad f_m f_{2i} = \sum_{j=0}^{i-1} f_{m+2i-2-4j},$$

$$m \geq 2i+1 : \quad f_m f_{2i+1} = f_{m-2i} + \sum_{j=0}^{i-1} f_{m+2i-1-4j}.$$

Multiplication Four Ways

- Freitag and Phillips (1998): Multiplication digit by digit, using

$$m \geq 2i: \qquad f_m f_{2i} = \sum_{j=0}^{i-1} f_{m+2i-2-4j},$$

$$m \geq 2i+1: \quad f_m f_{2i+1} = f_{m-2i} + \sum_{j=0}^{i-1} f_{m+2i-1-4j}.$$

Best to accumulate products and convert to Zeckendorf representation after every sum.

Multiplication Four Ways

- Freitag and Phillips (1998): Multiplication digit by digit, using

$$m \ge 2i: \qquad f_m f_{2i} = \sum_{j=0}^{i-1} f_{m+2i-2-4j},$$

$$m \ge 2i+1: \quad f_m f_{2i+1} = f_{m-2i} + \sum_{j=0}^{i-1} f_{m+2i-1-4j}.$$

  Best to accumulate products and convert to Zeckendorf representation after every sum.

- Tee (2002): Russian Peasant Multiplication: if y is even, $xy = (2x)(y/2)$, else $x + x(y-1) = x + (2x)((y-1)/2)$.

Multiplication Four Ways

- Freitag and Phillips (1998): Multiplication digit by digit, using

$$m \geq 2i: \qquad f_m f_{2i} = \sum_{j=0}^{i-1} f_{m+2i-2-4j},$$

$$m \geq 2i+1: \quad f_m f_{2i+1} = f_{m-2i} + \sum_{j=0}^{i-1} f_{m+2i-1-4j}.$$

  Best to accumulate products and convert to Zeckendorf representation after every sum.

- Tee (2002): Russian Peasant Multiplication: if y is even, $xy = (2x)(y/2)$, else $x + x(y-1) = x + (2x)((y-1)/2)$. Doubling: $1 \to 2$, return to Zeckendorf form.

Multiplication Four Ways

- Freitag and Phillips (1998): Multiplication digit by digit, using

$$m \geq 2i: \qquad f_m f_{2i} = \sum_{j=0}^{i-1} f_{m+2i-2-4j},$$

$$m \geq 2i+1: \quad f_m f_{2i+1} = f_{m-2i} + \sum_{j=0}^{i-1} f_{m+2i-1-4j}.$$

  Best to accumulate products and convert to Zeckendorf representation after every sum.

- Tee (2002): Russian Peasant Multiplication: if y is even, $xy = (2x)(y/2)$, else $x + x(y-1) = x + (2x)((y-1)/2)$. Doubling: $1 \to 2$, return to Zeckendorf form. Halving: left to right $xyz \to (x-1)(y+1)(z+1)$ when x is odd, at end digits are 0 or 2 apart from last, that may be 1.

Multiplication Four Ways

- Freitag and Phillips (1998): Multiplication digit by digit, using

$$m \geq 2i: \qquad f_m f_{2i} = \sum_{j=0}^{i-1} f_{m+2i-2-4j},$$

$$m \geq 2i+1: \quad f_m f_{2i+1} = f_{m-2i} + \sum_{j=0}^{i-1} f_{m+2i-1-4j}.$$

  Best to accumulate products and convert to Zeckendorf representation after every sum.

- Tee (2002): Russian Peasant Multiplication: if y is even, $xy = (2x)(y/2)$, else $x + x(y-1) = x + (2x)((y-1)/2)$. Doubling: $1 \rightarrow 2$, return to Zeckendorf form. Halving: left to right $xyz \rightarrow (x-1)(y+1)(z+1)$ when x is odd, at end digits are 0 or 2 apart from last, that may be 1. Then replace twos by ones.

Multiplication Continued

- Fenwick (2003): Egyptian multiplication successively doubles one number and by subtraction finds powers of two that make up the other number, adds appropriate powers.

Multiplication Continued

- Fenwick (2003): Egyptian multiplication successively doubles one number and by subtraction finds powers of two that make up the other number, adds appropriate powers. Requires remembering a list of numbers.

Multiplication Continued

- Fenwick (2003): Egyptian multiplication successively doubles one number and by subtraction finds powers of two that make up the other number, adds appropriate powers. Requires remembering a list of numbers.
  Fenwick replaced doubling by adding previous two: Fibonacci numbers instead of powers of two,

Multiplication Continued

- Fenwick (2003): Egyptian multiplication successively doubles one number and by subtraction finds powers of two that make up the other number, adds appropriate powers. Requires remembering a list of numbers.
  Fenwick replaced doubling by adding previous two: Fibonacci numbers instead of powers of two, adding instead of two doublings,

Multiplication Continued

- Fenwick (2003): Egyptian multiplication successively doubles one number and by subtraction finds powers of two that make up the other number, adds appropriate powers. Requires remembering a list of numbers.
  Fenwick replaced doubling by adding previous two: Fibonacci numbers instead of powers of two, adding instead of two doublings, but a bigger table.

Multiplication Continued

- Fenwick (2003): Egyptian multiplication successively doubles one number and by subtraction finds powers of two that make up the other number, adds appropriate powers. Requires remembering a list of numbers.
  Fenwick replaced doubling by adding previous two: Fibonacci numbers instead of powers of two, adding instead of two doublings, but a bigger table.
- Checkerboard: Napier multiplied on a checkerboard essentially using base two, as described in Gardner, "Knotted Doughnuts."

Multiplication Continued

- Fenwick (2003): Egyptian multiplication successively doubles one number and by subtraction finds powers of two that make up the other number, adds appropriate powers. Requires remembering a list of numbers.
  Fenwick replaced doubling by adding previous two: Fibonacci numbers instead of powers of two, adding instead of two doublings, but a bigger table.
- Checkerboard: Napier multiplied on a checkerboard essentially using base two, as described in Gardner, "Knotted Doughnuts." We can do the same in Zeckendorf form.

Conclusion

- Zeckendorf notation is an excellent technique for representing streams of variable length natural numbers, and is particularly good for continued fractions.

Conclusion

- Zeckendorf notation is an excellent technique for representing streams of variable length natural numbers, and is particularly good for continued fractions.
- Generalizing beyond Fibonacci numbers is possible, but turns out to not usually be useful.

Conclusion

- Zeckendorf notation is an excellent technique for representing streams of variable length natural numbers, and is particularly good for continued fractions.
- Generalizing beyond Fibonacci numbers is possible, but turns out to not usually be useful.
- Arithmetic on numbers in Zeckendorf form is straightforward, particularly with a checkerboard (division by Ancient Egyptian, Fibonacci adding or checkerboard).

Conclusion

- Zeckendorf notation is an excellent technique for representing streams of variable length natural numbers, and is particularly good for continued fractions.
- Generalizing beyond Fibonacci numbers is possible, but turns out to not usually be useful.
- Arithmetic on numbers in Zeckendorf form is straightforward, particularly with a checkerboard (division by Ancient Egyptian, Fibonacci adding or checkerboard).
- Future work: which multiplication/division algorithms are most efficient? Bunder (1992) Negafibonacci numbers for integers

Conclusion

- Zeckendorf notation is an excellent technique for representing streams of variable length natural numbers, and is particularly good for continued fractions.
- Generalizing beyond Fibonacci numbers is possible, but turns out to not usually be useful.
- Arithmetic on numbers in Zeckendorf form is straightforward, particularly with a checkerboard (division by Ancient Egyptian, Fibonacci adding or checkerboard).
- Future work: which multiplication/division algorithms are most efficient? Bunder (1992) Negafibonacci numbers for integers

Thank You