The Use and Usefulness of Fibonacci Codes
(*Invited talk*)

Shmuel T. Klein

Computer Science Department, Bar Ilan University, Israel
tomi@cs.biu.ac.il

1 Introduction

Contrary to our intuition led by the knowledge that the price for digital storage is constantly dropping, compression techniques are not becoming obsolete, and in fact research in data compression is flourishing as can be seen by the large number of papers published constantly on the topic. For instance, very large textual databases as those found in large Information Retrieval Systems, could contain hundreds of millions of words, which should be compressed by some method giving, in addition to good compression performance, also very fast decoding and the ability to search for the appearance of some strings directly in the compressed text.

Classical Huffman coding, when applied to individual characters, gives relatively poor compression, but when every word of a large textual database is considered as an atomic element to be encoded, this so-called Huffword variant may compete with the best other compression methods [11]. Yet the codewords of a binary Huffman code are not necessarily aligned on byte boundaries, which complicates both the decoding process and the ability to perform searches in the compressed file. The next step was therefore to pass to 256-ary Huffman coding, in which every codeword consists of an integral number of 8-bit bytes [4]. The loss incurred in the compression efficiency, which is only a few percent for large enough alphabets, is compensated for by the advantages of the easier processing.

When searches in the compressed text should also be supported, Huffman codes suffer from a problem of synchronization: denoting by $\mathcal{E}$ the encoding function, the compressed form $\mathcal{E}(x)$ of an element x may appear in the compressed text $\mathcal{E}(T)$, without corresponding to an actual occurrence of x in the text T, because the occurrence of $\mathcal{E}(x)$ is not necessarily aligned on codeword boundaries. This problem has been overcome in [8], relying on the tendency of Huffman codes to resynchronize quickly after errors, but the suggested solution is probabilistic and may produce wrong results. As alternative, [4] propose to reserve the first bit of each byte as *tag*, which is used to identify the last byte of each codeword, thereby reducing the order of the Huffman tree to 128-ary. These *Tagged Huffman codes* have then been replaced by *End-Tagged Dense codes* (ETDC) in [3] and by (s, c)-*Dense codes* (SCDC) in [2]. The two last mentioned codes consist of fixed codewords which do not depend on the probabilities of the items to be encoded. Thus their construction is simpler than that of Huffman codes: all one has to do is to sort the items by non-increasing frequency and then assign the codewords accordingly, starting with the shortest ones.

We show here that similar properties, and in fact some interesting others, can be obtained by *Fibonacci codes* [7], which have been suggested in the context of compression codes for the unbounded transmission of strings [1] and because of their robustness against errors in data communication applications [5]. They are also studied as a simple alternative to Huffman codes in [12]. The properties of representing

Shmuel T. Klein: The Use and Usefulness of Fibonacci Codes, pp. 1–5.
Proceedings of PSC 2016, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05996-8

integers in a Fibonacci based numeration system have been known long before the codes were suggested [13], and the following challenging quote appeared on page 211 in the book *Computer Number Systems and Arithmetic* by N.R. Scott in 1985:

> The Fibonacci number system (so-called by Knuth)
> is another remarkable and
> remarkably useless number system.

This obviously has been taken as an incentive to look for ever more useful applications of Fibonacci codes, not only as alternatives to dense codes for large textual word-based compression systems. They are in particular mentioned: in [10] as a good choice for compressing a set of small integers; in [6] to improve modular exponentiation; and in [9] as a basis to devise a rewriting code to enhance the repeated use of flash memory.

In the next section, we review the relevant features of Fibonacci codes of order $m \geq 2$. Examples of several applications will then be given in the talk itself.

2 Fibonacci codes

Fibonacci numbers of order $m \geq 2$, denoted by $F_i^{(m)}$, are defined by the following recurrence relation:

$$F_n^{(m)} = F_{n-1}^{(m)} + F_{n-2}^{(m)} + \cdots + F_{n-m}^{(m)} \qquad \text{for } n > 0,$$

and the boundary conditions

$$F_0^{(m)} = 1 \qquad \text{and} \qquad F_n^{(m)} = 0 \quad \text{for } -m < n < 0.$$

For fixed order m, the number $F_n^{(m)}$ can be represented as a linear combination of the nth powers of the roots of the corresponding polynomial $P(m) = x^m - x^{m-1} - \cdots - x - 1$. $P(m)$ has only one real root that is larger than 1, which we shall denote by $\phi_{(m)}$, the other $m-1$ roots are complex numbers with norm < 1 (for $m = 2$, the second root is also real and its absolute value is < 1). Therefore, when representing $F_n^{(m)}$ as such a linear combination, the term with $\phi_{(m)}^n$ will be the dominant one, and the others will rapidly become negligible for increasing n.

For example, $m = 2$ corresponds to the classical Fibonacci sequence and $\phi_{(2)} = \frac{1+\sqrt{5}}{2} = 1.6180$ is the well-known golden ratio. As a matter of fact, the entire Fibonacci sequence can be obtained by $F_n^{(m)} = [a_{(m)}\phi_{(m)}^n]$, where $a_{(m)}$ is the coefficient of the dominating term in the above mentioned linear combination, and $[x]$ means that the value of the real number x is rounded to the closest integer. Table 1 lists the first few elements of the Fibonacci sequences of order up to 6. The column headed *General Term* brings the values of $a_{(m)}$ and $\phi_{(m)}$. For larger n, the numbers $a_{(m)}\phi_{(m)}^n$ are usually quite close to integers.

The standard representation of an integer as a binary string is based on a numeration system whose basis elements are the powers of 2. If B is represented by the k-bit string $b_{k-1}b_{k-2}\cdots b_1b_0$, then $B = \sum_{i=0}^{k-1} b_i 2^i$. But many other possible binary representations do exist, and those using the Fibonacci sequences as basis elements have some interesting properties. Let us first consider the standard Fibonacci numbers of order 2.

$F_n^{(m)}$	General Term	1	2	3	4	5	6	7	8	9	10	11	12	13
$m=2$	$0.7236\ (1.6180)^n$	1	2	3	5	8	13	21	34	55	89	144	233	377
$m=3$	$0.6184\ (1.8393)^n$	1	2	4	7	13	24	44	81	149	274	504	927	1705
$m=4$	$0.5663\ (1.9275)^n$	1	2	4	8	15	29	56	108	208	401	773	1490	2872
$m=5$	$0.5379\ (1.9659)^n$	1	2	4	8	16	31	61	120	236	464	912	1793	3525
$m=6$	$0.5218\ (1.9836)^n$	1	2	4	8	16	32	63	125	248	492	976	1936	3840

Table 1. Fibonacci numbers of order $m = 2, 3, 4, 5, 6$

Any integer B can be represented by a binary string of length r, $c_r c_{r-1} \cdots c_2 c_1$, such that $B = \sum_{i=1}^{r} c_i F_i^{(2)}$. The representation will be unique if one uses the following procedure to produce it: given the integer B, find the largest Fibonacci number $F_r^{(2)}$ smaller or equal to B; then continue recursively with $B - F_r^{(2)}$. For example, $45 = 34 + 8 + 3$, so its binary Fibonacci representation would be 10010100. As a result of this encoding procedure, there are never consecutive Fibonacci numbers in any of these sums, implying that in the corresponding binary representation, there are no adjacent 1s.

This property can be exploited to devise an infinite code whose set of codewords consists of the Fibonacci representations of the integers: to assure the code being uniquely decipherable (UD), each codeword is prefixed by a single 1-bit, which acts like a *comma* and permits to identify the boundaries between the codewords. The first few elements of this code would thus be $\{u_1, u_2, \ldots\}$ = {**1**1, **1**10, **1**100, **1**101, **1**1000, **1**1001,...}, where the separating 1 is put in boldface for visibility. A typical compressed text could be **1**1001**1**100**1**101**1**1**1**101, which is easily parsed as $u_6 u_3 u_4 u_1 u_4$. Though being UD, this is not a prefix code, so decoding may be somewhat more involved. In particular, the first codeword 11, which is the only one containing no zeros, complicates the decoding, because if a run of several such codewords appears, the correct decoding of the codeword preceding the run depends on the parity of the length of the run. Consider for example the encoded string 11011111110: a first attempt to parse it as 110 | 11 | 11 | 11 | 10 $= u_2 u_1 u_1 u_1 10$ would fail, because the tail 10 is not a codeword; hence only when trying to decode the fifth codeword do we realize that the first one is not correct, and that the parsing should rather be 1101 | 11 | 11 | 110 $= u_4 u_1 u_1 u_2$.

To overcome this problem, [1,5] suggest to reverse all the codewords, yielding the set $\{v_1, v_2, \ldots\}$ = {11, 011, 0011, 1011, 00011, 10011, ...}, which is a prefix code, since all codewords are terminated by 11 and this substring does not appear anywhere in any codeword, except at its suffix. In addition, we show below that having a reversed representation, with the bits corresponding to increasing basis elements running from left to right rather than as usual, is advantageous for fast decoding. Table 2 brings a larger sample of this set of codewords in the column headed Fib2. Note that the order of the elements is not lexicographic, e.g., 10011 precedes 01011.

The generalization to higher order seems at first sight straightforward: any integer B can be uniquely represented by the string $d_s d_{s-1} \cdots d_2 d_1$ such that $B = \sum_{i=1}^{s} d_i F_i^{(m)}$ using the iterative encoding procedure mentioned above. In this representation, there are no consecutive substrings of m 1s. For example, the representations of the integers 10, 11, 12 and 13 using $F^{(3)}$ are, respectively, 1011, 1100, 1101 and 10000. But simply adding now $m-1$ 1's as commas and reversing the strings does not yield a prefix code for $m > 2$, and in fact the code so obtained is not even UD. For example, for $m = 3$, the above numbers would give the codewords $\{v_{10}, \dots, v_{13}\} = \{110\mathbf{11}, 001\mathbf{11}, 101\mathbf{11}, 0000\mathbf{11}\}$, but the encoding of the fourth element of the sequence would be $v_4 = 001\mathbf{11}$, which is a prefix of v_{11}. The string 0011110111 could be parsed both as $00111 \,\big|\, 10111 = v_4 v_5$ and as $001111 \,\big|\, 0111 = v_{11} v_2$. The problem stems from the fact that for $m > 2$, there can be more than one leading 1 in the representation of an integer, so adding $m - 1$ 1s may give a string of up to $2m - 2$ consecutive 1s. The fact that a string of m 1s appears only as a suffix is thus only true for $m = 2$. To turn the sequence into a prefix code, the definition has to be amended as follows: the set Fib$m$ will be defined as the set of binary codewords of lengths $\geq m$, such that every codeword contains exactly one occurrence of the substring consisting of m consecutive 1s, and this occurrence is the suffix of every codeword. The first elements of these codes for $m \leq 4$ are given in Table 2. For $m = 2$, this last definition is equivalent to the one above based on the representation with basis elements $F_n^{(2)}$; for $m > 2$, only a subset of the corresponding numbers is taken. There is nevertheless a connection between the codewords and the higher order Fibonacci numbers: for $m \geq 2$, and $n \geq 0$, the code Fib$m$ consists of

$$F_n^{(m)} \quad \text{codewords of length} \quad n + m.$$

index	Fib2	Fib3	Fib4
1	11	111	1111
2	011	0111	01111
3	0011	00111	001111
4	1011	10111	101111
5	00011	000111	0001111
6	10011	100111	1001111
7	01011	010111	0101111
8	000011	110111	1101111
9	100011	0000111	00001111
10	010011	1000111	10001111
11	001011	0100111	01001111
12	101011	1100111	11001111
13	0000011	0010111	00101111
14	1000011	1010111	10101111
15	0100011	0110111	01101111
16	0010011	00000111	11101111
17	1010011	10000111	000001111
18	0001011	01000111	100001111
19	1001011	11000111	010001111
20	0101011	00100111	110001111
21	00000011	10100111	001001111
22	10000011	01100111	101001111
23	01000011	00010111	011001111
24	00100011	10010111	111001111
25	10100011	01010111	000101111
26	00010011	11010111	100101111
27	10010011	00110111	010101111
28	01010011	10110111	110101111
29	00001011	000000111	001101111
30	10001011	100000111	101101111
31	01001011	010000111	011101111
32	00101011	110000111	0000001111
33	10101011	001000111	1000001111
34	000000011	101000111	0100001111
35	100000011	011000111	1100001111

Table 2. Fibonacci codes of order $m = 2, 3, 4$

This is visualized in Table 2, where for each code, blocks of codewords of the same length are separated by horizontal lines. Within each such block of lengths $\geq m + 2$ for Fib$m$, the prefixes of the codewords obtained by removing the terminating string of 1s correspond to consecutive integers in the representation based on $F^{(m)}$. For decoding, the Fibonacci representation will thus be used to get the relative index within the block, to which the starting index of the given block has to be added.

Many of the features of Fibonacci codes are based on the following facts. To represent an integer n, more bits are needed in the Fibonacci than in the standard representation, since it is less dense. In fact, it can be shown that the number of bits needed for $m = 2$ is $\lfloor \log_{\phi_2}(\sqrt{5}n) - 1 \rfloor \simeq 1.4404 \log_2 n$. On the other hand, the probability of a 1-bit drops from $\frac{1}{2}$ to only $\frac{1}{2}\left(1 - \frac{1}{\sqrt{5}}\right) = 0.276$, and thus the average number of 1-bits is only $0.389 \log_2 n$ instead of $0.5 \log_2 n$. This can be exploited for many applications.

References

1. A. Apostolico and A. Fraenkel: *Robust transmission of unbounded strings using Fibonacci representations*. IEEE Trans. Inform. Theory, IT–33 1987, pp. 238–245.
2. N. R. Brisaboa, A. Fariña, G. Ladra, G. Navarro, and M. Esteller: *(s,c)-dense coding: an optimized compression code for natural language text databases*, in Proc. Symposium on String Processing and Information Retrieval SPIRE'03, vol. 2857, LNCS, 2010, pp. 122–136.
3. N. R. Brisaboa, E. L. Iglesias, G. Navarro, and J. R. Paramá: *An efficient compression code for text databases*, in Advances in Information Retrieval, 25th European Conference on IR Research, ECIR 2003, Pisa, Italy, April 14-16, 2003, Proceedings, 2003, pp. 468–481.
4. E. S. de Moura, G. Navarro, N. Ziviani, and R. A. Baeza-Yates: *Fast and flexible word searching on compressed text*. ACM Trans. Inf. Syst., 18(2) 2000, pp. 113–139.
5. A. S. Fraenkel and S. T. Klein: *Robust universal complete codes for transmission and compression*. Discrete Applied Mathematics, 64(1) 1996, pp. 31–55.
6. S. T. Klein: *Should one always use repeated squaring for modular exponentiation?* Inf. Process. Lett., 106(6) 2008, pp. 232–237.
7. S. T. Klein and M. K. Ben-Nissan: *On the usefulness of Fibonacci compression codes*. Comput. J., 53(6) 2010, pp. 701–716.
8. S. T. Klein and D. Shapira: *Pattern matching in Huffman encoded texts*. Inf. Process. Manage., 41(4) 2005, pp. 829–841.
9. S. T. Klein and D. Shapira: *Boosting the compression of rewriting on flash memory*, in Data Compression Conference, DCC 2014, Snowbird, UT, USA, 26-28 March, 2014, 2014, pp. 193–202.
10. D. A. Lelewer and D. S. Hirschberg: *Data compression*. ACM Comput. Surv., 19(3) 1987, pp. 261–296.
11. A. Moffat: *Word-based text compression*. Softw., Pract. Exper., 19(2) 1989, pp. 185–198.
12. R. Przywarski, S. Grabowski, G. Navarro, and A. Salinger: *FM-KZ: an even simpler alphabet-independent FM-index*, in Proceedings of the Prague Stringology Conference, Prague, Czech Republic, August 28-30, 2006, 2006, pp. 226–241.
13. E. Zeckendorf: *Représentation des nombres naturels par une somme des nombres de Fibonacci ou de nombres de Lucas*. Bull. Soc. Roy. Sci. Liège, 41 1972, pp. 179–182.