FIBONACCI SEQUENCES AND MEMORY MANAGEMENT

T. G. LEWIS
University of Southwestern Louisiana, Lafayette, Louisiana 70501
B. J. SMITH and M. Z. SMITH
IBM Corporation, General Products Division, San Jose, California 95100

Fibonacci sequences have been studied from many points of view. We shall be concerned with sequences of integers which satisfy difference equations of the form

(1) $$L_i = L_{i-1} + L_{i-k-1} .$$

For various values of k, we obtain generalized Fibonacci sequences as studied by Daykin [1] and Hoggatt [4].

In [4], many interesting properties of these sequences are derived by using generating functions and generalized diagonals of Pascal's Triangle. For our purposes, however, we need a formula which allows the direct calculation of any particular term in any one of the sequences determined by (1). The techniques are standard (see Miles [6] or Flores [2]) but will be developed here for completeness.

The advantages of closed-form formulas are important to many applications of Fibonacci sequences. In particular, the solution of (1) is useful to computer scientists in their study of algorithms. The polyphase sort algorithm, for instance, requires the use of Fibonacci numbers, and Fibonacci numbers arise naturally in the analysis of the algorithm to compute the greatest common divisor of two numbers. The application we investigate concerns the way Fibonacci numbers can be used to manage computer memory.

Consider the objective of keeping as many jobs in memory as possible. To implement this, the system must keep extensive tables of areas in memory and the size of each area. As jobs finish, the memory area becomes checkerboarded with vacant blocks of various sizes. Sometimes a new job is a little too big to fit in any of these areas, even though the total available area is sufficient to accomodate several new jobs. This checkerboarding is referred to as *external fragmentation* of memory. It can be alleviated by rearranging the jobs in memory so that all the vacant space is in one place. However, such operations require computer resources which could otherwise be used for user jobs in memory.

Some systems arbitrarily divide memory into blocks of fixed size, and force the requests for memory space to conform to these constraints. This makes it more economical for the system to manage the available blocks and their locations. On the other hand, this can be extravagant use of memory, because requests for space seldom fill the blocks to which they are assigned. The unused memory area toward the end of these blocks is called *internal fragmentation.*

In fact, there are memory management schemes which incorporate some of these features [7]. One such system is the Buddy System [5] and works as follows. The total memory size m is a power of 2, say $m = 2^n$. when the system notes a request for storage space, it tries to find the smallest block still a power of 2, which will hold the request. Larger blocks may be split in half if available, creating two smaller blocks, either of which might hold the request with less wasted space (internal fragmentation).

One feature of this system which reduces system overhead results from the fact that each block size (there are n distinct sizes) is twice the size of the next smaller block. That is the block sizes L_i satisfy the relation $L_i = 2L_{i-1}$. If it happens that two adjacent blocks of size L_{i-1} become free, they are recombined into one block of size L_i. This makes the tables and search procedures somewhat simpler.

Others ([3], [5]) have noticed that the Buddy System equation is a special case of the more general difference equation:

(1) $$L_i = L_{i-1} + L_{i-k-1}, \qquad k = 0, 1, 2, \cdots .$$

For $k = 0$ (and appropriate initial values) we get the sequence 1, 2, 4, 8, 16, $\cdots$. For $k = 1$, we can obtain the Fibonacci

sequence 1, 1, 2, 3, 5, 6, 13, ··· . For other values of k we will refer to the corresponding sequences as the k^{th} Fibonacci sequence (see Table 1).

Table 1
Generalized Fibonacci Sequences Giving Block Sizes 1 Through 250 (approx.)

i	L_i				
Level	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
1	1	1	1	1	1
2	2	1	1	1	1
3	4	2	1	1	1
4	8	3	2	1	1
5	16	5	3	2	1
6	32	8	4	3	2
7	64	13	6	4	3
8	128	21	9	5	4
9	256	34	13	7	5
10		55	19	10	6
11		89	28	14	8
12		144	41	19	11
13		233	60	26	15
14			88	36	20
15			129	50	26
16			189	69	34
17			277	95	45
18				131	60
19				181	80
20				250	106
21					140
22					185
23					245

We can design a memory management scheme based on the k^{th} Fibonacci sequence, modeled after the Buddy System. Initially, memory is the size of an appropriate Fibonacci number, and requests for smaller pieces of memory are serviced by using Eq. (1) to split and reassemble blocks. The question is, does this improve utilization of memory? Table 1 shows there is a greater variety of block sizes as k increases. We conjecture that internal fragmentation decreases as k increases, but that system overhead increases.

For the moment we will disregard the overhead and examine the cost due to internal fragmentation. Let $\{L_i\}_{i=0}^{n}$ be the collection of block sizes, with $L_0 = 0$ and $L_n = m$ the memory size. If the system services a request for a certain number x of memory locations, it will allocate a block of size L_i, where $L_{i-1} < x \leq L_i$. The waste involved is $(L_i - x)$.

The requests for memory space are always for an integral number of locations, but for convenience let us assume that the request sizes are given by a continuous probability function $pdf(x)$. Then the expected average waste per request $\bar{w}$ is given by Hirschberg in [3]:

$$\bar{w} = \sum_{i=1}^{n} \int_{L_{i-1}}^{L_i} (L_i - x) pdf(x) dx .$$

Rewriting this, we obtain

$$\bar{w} = m - \bar{x} - \sum_{i=1}^{n} (d_i) cdf(L_{i-1}) , \tag{2}$$

where:

m = maximum memory size

$\bar{x} = \int_0^m x\,pdf(x)$ = avg. request size

n = number of distinct block sizes

$d_i = L_i - L_{i-1}$

$cdf(z) = \int_0^z pdf(z)dx$ = cumulative distribution function.

The objective of memory management is to minimize $\bar{w}$ for a given $pdf(x)$. If we restrict our attention to Fibonacci type systems, we can gain some additional insight into minimizing $\bar{w}$.

Equation 1 gives rise to the characteristic polynomial, $x^{k+1} - x^k - 1 = 0$, of the k^{th} Fibonacci sequence. The polynomial (for fixed k) is known to have $(k+1)$ distinct roots which yield a closed-form expression for the n^{th} Fibonacci number. Note that $f(x) = x^{k+1} - x^k - 1$ has a real root between 1 and 2, since $f(1)$ is negative and $f(2)$ is positive. By Descartes' rule of signs, this is the only positive root, which will be denoted by a_1. Thus, $1 < a_1 \leqslant 2$ ($a_1 = 2$ for $k = 0$). Let the other roots of $f(x)$ be $a_2, a_3, \cdots, a_{k+1}$. It is easy to establish that a_1 is the root of largest modulus and, in fact, $|a_i| < 1$ for $i = 2, 3, \cdots, k+1$. (See, for example, [8].)

Evidently, any sequence of numbers $\{u_i\}$ satisfying

$$u_i = c_1 a_1^i + c_2 a_2^i + \cdots + c_{k+1} a_{k+1}^i$$

will be a k^{th} Fibonacci sequence satisfying Eq. (1). Specifying the initial $(k+1)$ terms of the sequence determines the constants $c_1, \cdots, c_{k+1}$, or specifying the constants determines the sequence. For the particular sequence $\{L_i\}$ in Table 1, we can write

$$L_i = c_1 a_1^i + \cdots + c_{k+1} a_{k+1}^i .$$

Since $|a_i| < 1$ for $i = 2, 3, \cdots, k+1$, it follows that for sufficiently large i,

$$L_i \cong c_1 a_1^i .$$

Some approximate values of c_1 and a_1 are given in Table 2. The initial segments in Table 1 can be obtained from the formula $L_i = c_1 a_1^i$ (rounded to the nearest integer).

Table 2
Generators for Fibonacci Sequences

	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$c(k)$	1	.44721	.41724	.39663	.38119
$a(k)$	2	1.61803	1.46557	1.38028	1.32472

Consider now the value of a_1 for different values of k. Let this root be denoted by $a(k)$. We have observed that $1 < a(k) \leqslant 2$. From Eq. (1) we see that

$$a(k) = 1 + \left(\frac{1}{a(k)}\right)^k ,$$

and it follows that $a(k+1) < a(k)$ for every k. In fact, $\lim_{k \to \infty} a(k) = 1$

Let us apply the preceding observations to a particular example, in which the distribution of request sizes is given by the uniform distribution $pdf(x) = (1/m)$. Then $cdf(x) = (x/m)$, and $\bar{x} = (m/2)$. Let k be arbitrary but fixed. We write $a(k) = a$ and $c(k) = c$, so that $L_i \cong ca^i$, and $L_n = ca^n = m$. Then

$$\bar{w} = m - \bar{x} - \sum_{i=1}^{n} (d_i)cdf(L_{i-1}) = m - \frac{m}{2} - c\left(1 - \frac{1}{a}\right)\sum_{i=1}^{n} a^i cdf(ca^{i-1}) \tag{3}$$

$$= \frac{m}{2} - c\left(1 - \frac{1}{a}\right)\sum_{i=1}^{n} a^i \frac{ca^i}{a^m} \quad .$$

If we assume that $m >> 1$, then

$$a^2m^2 - 1 \cong a^2m^2, \qquad \text{and} \qquad \bar{w} \cong \frac{m}{2} - \frac{m}{a+1} \quad .$$

Thus, $\bar{w}$ can be made as small as desired by increasing k, since a approaches 1 as k increases.

Intuitively, this is to be expected, since for any finite memory size m, if $k > m$, then the k^{th} Fibonacci sequence contains all the integers from 1 through m, and $\bar{w}$ should be zero. However, this leads to extreme overhead in memory management and places unreasonable demands on the search mechanism for allocation and release of area in memory.

The waste function $\bar{w}$ measures only the cost of internal fragmentation. Let us assume that the overhead associated with a memory system is given by a function of n, the number of distinct block sizes. Then a more complete cost function is

$$w = m - \bar{x} - \sum_{i=1}^{n} (d_i)cdf(L_{i-1}) + f(n).$$

This raises the possibility of optimizing the collection $\{L_i\}_{i=0}^{n}$ by considering the equations

$$\frac{\partial w}{\partial L_j} = 0 \quad \text{for} \quad j = 1, 2, \cdots, n-1$$

and the boundary conditions $L_0 = 0$, $L_n = m$. The solution is given by

$$L_{j+1} = L_j + \frac{cdf(L_j) - cdf(L_{j-1})}{cdf(L_j)} \quad . \tag{4}$$

Continuing with the simple example of the uniform request distribution, let us assume conveniently that $f(n) = \beta \cdot n$, where $\beta > 0$ is a constant. We obtain

$$L_{j+1} = L_j + \frac{\frac{L_j}{m} - \frac{L_{j-1}}{m}}{\frac{1}{m}} = 2L_j - L_{j-1} \ .$$

The difference equation is not of the Fibonacci type, but does have a closed form solution

$$L_j = \frac{m}{n} j, \qquad j = 0, 1, \cdots, n.$$

So it is possible to optimize the collection $\{L_j\}$, which minimizes w, provided we know the nature of $f(n)$ in Eq. (4).

Unfortunately, other request distributions and other functions $f(n)$ do not lead to such nice solutions. Indeed the difference equations resulting from (4) are, in general, extremely difficult to solve analytically. For certain $pdf(x)$'s, however, solutions are of considerable importance to computer systems designers, and where closed-form solutions of the difference equations are not feasible, it is still important to apply numerical techniques to these problems.

REFERENCES

1. D. E. Daykin, "Representation of Natural Numbers as Sums of Generalized Fibonacci Numbers," *Journal London Math. Soc.*, 35 (1960), pp. 143–160.

2. I. Flores, "Direct Calculation of k Generalized Fibonacci Numbers," *The Fibonacci Quarterly,* Vol. 5, No. 3 (Apr. 1967), pp. 259–266.
3. D. S. Hirschberg, "A Class of Dynamic Memory Allocation Systems," *Comm. ACM,* 16, 19 (Oct. 1973), pp. 615–618.
4. V. E. Hoggatt, Jr., "A New Angle on Pascal's Triangle," *The Fibonacci Quarterly,* Vol. 6, No. 4 (Dec. 1968), pp. 221–234.
5. D. E. Knuth, *The Art of Computer Programming,* Vol. I (2nd Ed.), Addison-Wesley, Reading, Mass., 1973, pp. 78–96, 435–455.
6. E. P. Miles, "Generalized Fibonacci Numbers and Associated Matrices," *Amer. Math. Monthly,* 67 (1967), pp. 745–757.
7. J. Minker, et al., "Analysis of Data Processing System," Tech. Rept. 69–99, University of Maryland, College Park, Md., 1969.
8. B. T. Smith, "Error Bounds for Zeros of a Polynomial Based on Gerschgorin's Theorem," *J. ACM,* 17, 4 (Oct. 1970), pp. 661–674.

****

[Continued from Page 29.]

$$\begin{array}{rcl} 89+11 = 100 & \begin{matrix}>35\\>56\\>90\\>145\end{matrix} & >21 \\ 144+12 = 156 & & >34 \\ 233+13 = 246 & & >55 \\ 377+14 = 391 & & \\ \text{etc., etc., etc.} & & \end{array} \qquad \begin{array}{rcl} 89-11 = 78 & \begin{matrix}>33\\>54\\>88\\>143\end{matrix} & >21 \\ 144-12 = 132 & & > \\ 233-13 = 220 & & >55 \\ 377-14 = 363 & & \\ \text{etc., etc., etc.} & & \end{array}$$

Now try it with the Lucas series 1, 3, 4, 7, 11, ⋯.

N.B–(In the reverse Fibonacci sequence, F_n is negative for even n).

****