OPTIMAL COMPUTATION, BY COMPUTER, OF FIBONACCI NUMBERS

Arie Rokach
Kedumim 44856, Israel
(*Submitted February 1995*)

1. INTRODUCTION

For a given index n, this paper presents an alternative way of computing F_n. Instead of using addition only, F_n. can be computed using addition, multiplication, and formula (2) below. The formula has a parameter k, and this paper finds such k that minimizes the number of arithmetic operations required to calculate F_n. In this paper, it is assumed that n is large enough that F_n is not stored in a single computer word but, rather, is represented as an array of BITS.

2. BASIC PROPERTIES

In this section we present some relevant features of Fibonacci numbers.

Lemma 1:

$$F(i,j,k) = L_k * F(i,j-1,k) + (-1)^{(k+1)} * F(i,j-2,k), \tag{1}$$

where $F(i,j,k)$ denotes the value of the Fibonacci number that occupies the (i,j) position of a matrix when the Fibonacci numbers are arranged, k elements in a column, or

$$F_{n+k} = F_n * L_k + (-1)^{k+1} * F_{n-k}, \tag{2}$$

where k denotes the number of elements in one such column, L_k represents the Lucas numbers, and n is arbitrary.

Proof: From [1], we have

$$F_n = (\alpha^n - \beta^n)/\sqrt{5}, \tag{3}$$

where $\alpha = (1+\sqrt{5})/2$ and $\beta = (1-\sqrt{5})/2$. It is obvious that $\beta = -1/\alpha$; thus,

$$F_n = (\alpha^n - (-1)^n * \alpha^{-n})/\sqrt{5}. \tag{4}$$

Also, it is known that

$$L_k = (\alpha^k + \beta^k) = \alpha^k + (-1)^k * (\alpha)^{-k}. \tag{5}$$

Therefore,

$$F_{n+k} = (\alpha^{n+k} - (-1)^{n+k} * \alpha^{-n-k})/\sqrt{5}, \tag{6}$$

$$\begin{aligned} F_n * L_k + (-1)^{k+1} * F_{n-k} &= (\alpha^{n+k} - (-1)^n * \alpha^{k-n} + (-1)^k \alpha^{n-k} - (-1)^{n+k} \alpha^{-n-k} \\ &\quad - (-1)^k \alpha^{n-k} + (-1)^{n-k+k} \alpha^{-(n-k)})/\sqrt{5} \\ &= (\alpha^{n+k} - (-1)^{n+k} * \alpha^{-n-k})/\sqrt{5} = F_{n+k}. \end{aligned} \tag{7}$$

Q.E.D.

The importance of Lemma 1 is that F_n can now be calculated by using multiplication along with addition. In order to minimize the effort of calculation, we have to calculate the addition effort and the multiplication effort. This is done in Lemmas 2 and 3.

Lemma 2: $O(X+Y) = \delta * MIN(LOG_2(X), LOG_2(Y))$, where X and Y are arrays of binary digits, δ is at most 7, and $O(X+Y)$ denotes the number of arithmetic operations that are performed when adding X to Y.

Proof: The addition algorithm is:

```
CARRY = 0.
    For i = 1 to MIN(LOG_2(X), LOG_2(Y)),
    TEMP = (X(i) XOR Y(i)) XOR CARRY            (2 operations)      (8)
    CARRY = (X(i) AND Y(i)) OR (X(i) AND CARRY)                     (9)
              OR (Y(i) AND CARRY)               (5 operations)
    Y(i) = TEMP
    END
```

At the end of the process, the result is found in Y. The number of operations in the loop (8), (9) is 7. This will be referred to later as δ. The loop is executed $MIN(LOG_2(X), LOG_2(Y))$ times. This immediately proves Lemma 2. Q.E.D.

Lemma 3: $O(X*Y) = \delta * LOG_2(X) * (One(Y) - 1)$, where X and Y are arrays of binary digits, One(X) denotes the number of 1's in the binary representation of X, and $O(X*Y)$ is the number of operations that are performed when multiplying X by Y.

Proof: The following algorithm performs the multiplication:

```
Set Z to ZERO     (Z is the result array)
Set SHIFT to ZERO
For i = 1 to LOG_2(Y)
    WHILE Y(i) = 0
        i = i + 1
        SHIFT = SHIFT + 1
    END
    IF SHIFT > 0
        ADD(X, Z, SHIFT)     [see (12)]                             (10)
        SHIFT = SHIFT +1                                            (11)
    END
END
```

ADD(X, Z, SHIFT) is an addition algorithm that adds X to Z, (12)

such that X(1) is added to Z(SHIFT), X(2) is added to Z(SHIFT + 1), etc. The result is stored in Z.

Since the main arithmetic effort of the multiplication (10) is executed (One (Y) −1) times, and each execution "costs," according to Lemma 2, $\delta * \mathrm{LOG}_2(\mathrm{X})$ operations. The number of operations is $O(\mathrm{X} * \mathrm{Y}) = \delta * \mathrm{LOG}_2(\mathrm{X}) * (\mathrm{One}(\mathrm{Y}) - 1)$. Q.E.D.

Lemma 4 computes $\log_2(F_n)$ while Conjecture 1 computes $\mathrm{One}(L_k)$.

Lemma 4: $\mathrm{Log}_2(F_n) = (\log_2 \alpha) * n$, where $\alpha = (1+\sqrt{5})/2$.

Later, we shall refer to $\log \alpha$ as 2λ (approximately 0.69).

Proof: $F_n = {\sim\sim}\ \alpha ** n/\sqrt{5}$ ([1], p. 82; Eq. (15) below); $\log_2(F_n) = {\sim\sim}\ \log_2 \alpha * n = 2\lambda * n$. Q.E.D.

Conjecture 1: $\mathrm{One}(L_n) = {\sim\sim}\ \lambda n$. For example:

n	1192	1193	1194	1195	1196	1197	1198	1199	1200
$\mathrm{One}(L_n)$	402	429	408	412	437	448	435	417	406

This paper claims that the conjecture is true, without proof; it was found by calculating F_n for large numbers using a computer. We can see that $\mathrm{One}(L_n)$ has "Ups" and "Downs" but, in general, it fits the above formula.

2. MINIMIZING $O(F_n)$

Assume, for simplicity, that $n = k * \ell$, $\ell = n/k$.* According to (2), the following algorithm computes F_n:

Compute F_0 up to F_{k+1}. In this way, L_k is already computed.

[*The required number of operations for this computation is*

$$\sum_{j=1}^{k} 2\lambda\delta j = \lambda\delta(k^2+k)$$

(see Lemma 2 and Lemma 4).]

For $i = 1$ to $\ell - 1$,

Temp = $F_{ik} * L_k$.

[*The required number of operations for this computation is*

$$2k^2\lambda^2\delta\sum_{i=1}^{\ell-1} i - 2k\lambda\delta\sum_{i=1}^{\ell-1} i = \lambda^2\delta n(n-k) - \lambda\delta n(n-k)/k$$

(see Lemma 3 and Conjecture 1)].

* Although the setting $n = k * \ell$ seems arbitrary, it was found by running a computer program that the optimal k is (1/6)n, which is very close to our result $\lambda n/2 = {\sim\sim}\ 0.17n$.

$F_{(i+1)k} = \text{Temp} + (-1)^{k+1} * F_{(i-1)k}$ [see (2)].

[*The required number of operations for this calculation is*

$$\delta\sum_{i=1}^{\ell-1}\log_2 \text{Temp} = \delta\sum_{i=1}^{\ell-1}\log_2 F_{ik} + \delta\sum_{i=1}^{\ell-1}\log_2 L_k$$
$$= \delta\lambda n(n-k)/k + \delta\lambda k(n-k)/k$$
$$= \delta\lambda n(n-k)/k + \delta\lambda n - \delta\lambda k.]$$

END.

The total number of operations is given by

$$O(F_n) = \lambda\delta(k^2+k) + \lambda^2\delta n(n-k) + \delta\lambda n - \lambda\delta k. \tag{13}$$

In order to find the optimal k, we derive, by (13),

$$\lambda\delta(2k+1) - \lambda^2\delta n - \lambda\delta = 0;$$
$$k = \lambda n \backslash 2. \tag{14}$$

3. COMPARISON

Substituting the optimal $k:\lambda n\backslash 2$ in (13) yields

$$\begin{aligned} O(F_n) &= \lambda\delta(\lambda^2 n^2/4) + \lambda^2\delta n(n-\lambda n/2) + \delta\lambda n - \lambda\delta n^2/2 \\ &= \delta\lambda^3 n^2/4 + \delta\lambda^2 n/2 + \boldsymbol{\delta\lambda^2 n^2} - \delta\lambda^3 n/2 + \delta\lambda n - \delta\lambda^2 n/2 \\ &=^{\sim\sim} \boldsymbol{\delta\lambda^2 n^2}, \end{aligned} \tag{15}$$

where the bold terms represent the significant terms.

On the other hand, the computation of F_n in the regular way needs

$$\begin{aligned} O(F_n) &= 2\delta\lambda(1+2+3+\cdots+n-1) \\ &= 2\delta\lambda(1+n-1)(n-1)/2 \\ &=^{\sim\sim} \boldsymbol{\delta\lambda n^2} \end{aligned} \tag{16}$$

(see Lemma 2).

The ratio between the expression in (15) and that in (16) is λ. This means that the proposed method is $1/\lambda$ (approximately 2.88) times more efficient than the regular method.

REFERENCE

1. D. Knuth. *The Art of Computer Programming*. Vol. 1. New York: Addison Wesley, 1969.

AMS Classification Number: 11Y55

❖❖❖