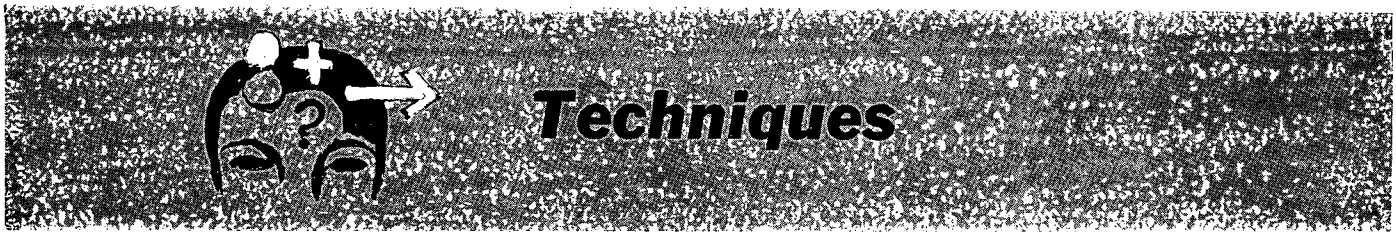


R. W. BEMER, Editor

Evaluation of Polynomials By Computer

DONALD E. KNUTH
California Institute of Technology
Pasadena, California

In many applications it becomes necessary to evaluate

$$P = y^n + a_1y^{n-1} + a_2y^{n-2} + \cdots + a_n,$$

where the a's are constants known to the programmer. For example, one may wish to compute

$$P = y^4 + 3y^3 + 5y^2 + 7y + 9.$$

A beginner would tend to write this as

$$P := y\uparrow 4 + 3 \times y\uparrow 3 + 5 \times y\uparrow 2 + 7 \times y + 9.$$

But experienced programmers seeing this would smile and show him the "professional" way to compute it:

$$P := (((y+3)\times y+5)\times y+7) \times y + 9.$$

This ingenious way of rewriting the expression (sometimes called Horner's Rule) needs only 3 multiplications and 4 additions, while the previous way of writing it requires 8 multiplications (or even 9, depending on how $y\uparrow 4$ is evaluated) and 4 additions. Recently Motzkin [1] showed that for certain polynomials there is yet a better way to do the evaluation, and his discovery inspired the study which led to this paper.

The above polynomial can, for example, be calculated using the sequence

$$z := y \times (y+1); \quad P := (z+y-1) \times (z+4) + 13.$$

Here 2 multiplications and 5 additions are used, so a multiplication has been traded for an addition. Using floating-point hardware, a multiplication will take perhaps 5 times as long as an addition. In fixed-point arithmetic (for example in the common case of a sine or arctangent subroutine), a multiplication will often take up to 20 or more times as long as an addition. However, if the calculations are done by floating-point subroutines on a machine which has no floating-point hardware, an addition often takes roughly as long as a multiplication; so nothing has been gained. We will show, however, that for degree 5 or more the *total* number of operations, counting additions and multiplications with equal weight, can even be reduced. Specifically, we will show that an arbitrary polynomial of degree n requires multiplications and additions as follows:

n	$\times$	$+$
4	2	5
5	3	5
6	3	7
7	4	8
8	5, usually 4	9

Preliminary Considerations

The method given in [1] shows how a sixth degree polynomial can be computed with 3 multiplications and 7 additions. Horner's rule takes $n-1$ multiplications and n additions; so this saves two multiplications at the expense of one extra addition. Unfortunately, however, that method requires the solving of a quadratic equation, and this quadratic might not have any real roots. There are, in fact, many sixth degree polynomials which cannot be handled by that algorithm, and it seems to require no less than a major modification to make it cover all cases. One would of course expect to be able to handle *special* polynomials in a more efficient manner than a general polynomial; and any method will work for certain conditions on the coefficients of the polynomial. So this leaves open the question whether *every* sixth degree polynomial can be done with 3 multiplications. While trying to prove this impossible, the author discovered that it was indeed possible.

First, we observe that the general fourth degree equation can always be evaluated by

$$z := y \times (y+a); \quad P := (z+y+b) \times (z+c) + d.$$

Treating the polynomial as

$$P = y^4 + Ay^3 + By^2 + Cy + D,$$

one way to evaluate a, b, c and d is:

$$\begin{aligned} a &= \frac{A-1}{2} \\ b &= B(a+1) - C - a(a+1)^2 \\ c &= B - b - a(a+1) \\ d &= D - bc \end{aligned}$$

Sixth Degree Polynomials

Moving to degree six, we wish to evaluate

$$P = y^6 + Ay^5 + By^4 + Cy^3 + Dy^2 + Ey + F.$$

This can always be done by:

$$z := y \times (y+a);$$
$$w := (z+b) \times (y+c)$$
$$P := (w+z+d) \times (w+e) + f$$

The calculation of a, b, c, d, e and f is much more trouble this time, but it is worthwhile if the polynomial is to be evaluated many times as in a library subroutine. A relatively simple computer program can be written to calculate the new constants; a good compiler might also even do such calculation while compiling.

We start by solving the following set of equations, into which new variables p, q, r and s are introduced:

$$2p + 1 = A \tag{1}$$

$$p(p+1) + 2q + a = B \tag{2}$$

$$p(2q+a) + q + r + s = C \tag{3}$$

$$p(r+s) + r + q(q+a) = D \tag{4}$$

$$ar + q(r+s) = E \tag{5}$$

Equation (1) determines p, and then equation (2) determines a as a linear function of q. Equation (3) determines $r+s$ as another linear function of q, and using this fact in (4) we have r as a quadratic in q. Then equation (5) becomes a cubic equation in q. More precisely, we can obtain the modified equations:

$$a = B' - 2q \tag{2'}$$

$$r + s = C' - q \tag{3'}$$

$$r = q^2 + D'q + D'' \tag{4'}$$

$$2q^3 + E'q^2 + E''q + E''' = 0. \tag{5'}$$

where:

$$B' = B - p(p+1)$$
$$C' = C - pB'$$
$$D' = p - B'$$
$$D'' = D - pC'$$
$$E' = 2D' - B' + 1$$
$$E'' = 2D'' - B'D' - C'$$
$$E''' = E - B'D''$$

Equation (5'), being a cubic equation, always has a real root q. Once q is known we can find a, r, s by the above equations and then finish by:

$$c = p - a \tag{6}$$

$$b = q - ac \tag{7}$$

$$d = s - bc \tag{8}$$

$$e = r - bc \tag{9}$$

$$f = F - rs \tag{10}$$

Example. Suppose we are given the polynomial

$$P = y^6 + 13y^5 + 49y^4 + 33y^3 - 61y^2 - 37y + 3,$$

we find $p = 6$ and obtain the cubic equation

$$2q^3 - 8q^2 + 2q + 12 = 0.$$

This cubic has the roots 2, 3 and -1, and these lead to $(r, s) = (-5, -6)$, $(-1, -11)$, and $(-5, -3)$, respectively. There are therefore three equally good ways to compute the given polynomial:

$$z := y \times (y + 3)$$
$$w := (z - 7) \times (y + 3)$$
$$P := (w + z + 15) \times (w + 16) - 27$$

$$z := y \times (y + 1)$$
$$w := (z - 2) \times (y + 5)$$
$$P := (w + z - 1) \times (w + 9) - 8$$

$$z := y \times (y + 9)$$
$$w := (z + 26) \times (y - 3)$$
$$P := (w + z + 75) \times (w + 73) - 12.$$

The first solution shown is actually a little better, since the computation of $y+3$ needs to be performed only once.

Arbitrary Polynomials of Even Degree

We now develop a different algorithm for arbitrary degrees. This general method will not always be the best possible but it gives best possible results for a large class of polynomials (more than 50%) of degree 6, 7 or 8 and probably of higher orders as well. Let the given polynomial

$$P = y^n + a_1y^{n-1} + a_2y^{n-2} + \cdots + a_n, \qquad n \geqq 3.$$

First suppose $n = 2m$ is even. Then P can be written as

$$z^n + z^{n-1} + b_2z^{n-2} + \cdots + b_n,$$

where z is[1] $y+t$, $t = (a_1-1)/n$.

The next step is the crucial one—to try to solve the $(m-1)$st degree equation

$$\alpha^{m-1} + b_3\alpha^{m-2} + b_5\alpha^{m-3} + \cdots + b_{n-1} = 0.$$

Call this equation a *reduction* equation.

If this polynomial has a real root α, we can apply a squaring rule,

$$P = (z^{n-2}+z^{n-3}+c_2z^{n-4}+\cdots+c_{n-2})(z^2-\alpha_n) + \beta_n$$

where:

$$c_2 = b_2 + \alpha \qquad c_3 = b_3 + \alpha$$
$$c_4 = b_4 + c_2\alpha \qquad c_5 = b_5 + c_3\alpha$$
$$c_6 = b_6 + c_4\alpha \qquad c_7 = b_7 + c_5\alpha$$
$$\vdots \qquad\qquad \vdots$$
$$c_n = b_n + c_{n-2}\alpha \qquad c_{n-3} = b_{n-3} + c_{n-5}\alpha$$
$$\beta_n = c_n \qquad\qquad \alpha_n = \alpha$$

As a check we can calculate $c_{n-1} = b_{n-1} + c_{n-3}\alpha = 0$.

If the polynomial has no real root this is a shame, and

[1] Another algorithm which is very similar can be used with $t = (a_1 + 1)/n$. This modification sometimes leads to fewer multiplications, e.g. with $x^6 + x^5 + x$. Also, it might give even greater accuracy if a_1 is nearly equal to 1, although this depends on the other conditions.

we apply Horner's rule twice to reduce the degree,

$$P = ((z^{n-2}+z^{n-3}+c_2z^{n-4}+\cdots+c_{n-2})z+\alpha_n)z + \beta_n .$$
$$c_i = b_i , \quad \alpha_n = b_{n-1} , \quad \beta_n = b_n .$$

The same process is now iterated on the reduced polynomial.

Eventually we will have reduced n to 2, and we will have

$$P' = (z^2+z+c)(z^2-\alpha_4) + \beta_4 ,$$

which is easy to evaluate.

The final algorithm is then

$z := y + t;$
$w := z \uparrow 2;$
$P1 := w + z + c;$
$P2 := P1\times(w-\alpha_4) + \beta_4$ or $(P1\times z+\alpha_4)\times z+\beta_4$;
$P3 := P2\times(w-\alpha_6) + \beta_6$ or $(P2\times z+\alpha_6)\times z+\beta_6$;

etc., the choices in the latter steps depending on whether the reduction equation was solvable or not. (The reduction equations for $P2$, $P4$, $P6$, etc. are of odd degree. Thus they are always solvable, and the "choice" implied by the word "or" above is really nonexistent.) This gives us approximately $n/2$ multiplications and $n+1$ additions. Actually there are $n-r-1$ multiplications, where r is the number of reduction equations we were able to solve.

The algorithm just described is equivalent to Motzkin's for $n = 6$, but this presentation is more revealing as to the inner mechanism in that method. The algorithm for $n = 4$ is simply the one mentioned above.

Example.

$$P = z^8 + z^7 + 3z^6 + 2z^5 + 3z^4 - z^3 + 3z^2 - 2z + 1.$$

Here $t = 0$, so the reader can see immediately that the example is rigged. We try first to solve the first reduction equation, $\alpha^3 + 2\alpha^2 - \alpha - 2 = 0$, and take the root $\alpha = -2$; so

$$P = (z^6+z^5+z^4+z^2-z+1)(z^2+2) - 1.$$

The next reduction equation is $\alpha^2 - 1 = 0$ and we take $\alpha = -1$; thus

$$P = ((z^4+z^3-z+1)(z^2+1)+0)(z^2+2) - 1.$$

The final reduction equation is $\alpha - 1 = 0$; so we have

$$P = (((z^2+z+1)(z^2-1)+2)(z^2+1)+0)(z^2+2) - 1.$$

Arbitrary Polynomials of Odd Degree

If n is odd, there are several alternatives. We could simply reduce the degree by one with one application of Horner's Rule and then apply the preceding method. Or we can reduce the degree by 2, using the squaring rule—a simple and straightforward modification of the squaring rule for even exponents. For example, if $n = 7$ we solve the cubic reduction equation $\alpha^3 + b_2\alpha^2 + b_4\alpha + b_6 = 0$. With this the problem can be reduced to that of evaluating a fifth degree equation. The fifth degree equation is reduced to degree 4 by using Horner's Rule. This shows that an arbitrary seventh degree polynomial can be computed using 4 multiplications and 8 additions, and the derivation of the constants is relatively simple. It seems that a good method to use would be to reduce n by 2 if $n = 4k - 1$, and to reduce by 1 if $n = 4k + 1$, thus avoiding the solution of at least one equation of even degree.

Another method is often useful when $n = 2m - 1$ is odd. With $P = y^n + a_1y^{n-1} + \cdots + a_n$ as before, we need not use the transformation $z = y + t$, but we can work with the original equation. Form the reduction equation $\alpha^{m-1} + a_2\alpha^{m-2} + a_4\alpha^{m-3} + \cdots + a_{n-1} = 0$. If α solves this equation the squaring rule can be used. By reducing n in steps of 2 it will eventually be reduced to a cubic polynomial evaluated in the form $(y+a_1)(y^2-\alpha) + \beta$. More precisely, the algorithm will be

$w := y \uparrow 2;$
$P1 := y + a_1 ;$
$P2 := P1 \times (w-\alpha_3) + \beta_3$
or $((P1\times y)+\alpha_3) \times y + \beta_3 ;$
$P3 := P2 \times (w-\alpha_5) + \beta_5$
or $((P2\times y)+\alpha_5) \times y + \beta_5 ;$

etc.

There are n additions instead of $n+1$, and $n-1-r$ multiplications.

For example, let $n = 5$.

$$P = y^5 + a_1y^4 + a_2y^3 + a_3y^2 + a_4y + a_5 .$$

Suppose we can solve $\alpha^2 + a_2\alpha + a_4 = 0$. Then we have

$$P = (y^3+a_1y^2+c_2y+c_3)(y^2-\alpha_5) + \beta_5$$

where

$$c_2 = a_2 + \alpha, \quad c_3 = a_3 + a_1\alpha, \quad \alpha_5 = \alpha, \quad \beta_5 = a_5 + c_3\alpha.$$

The final reduction equation is $\alpha + c_2 = 0$, which is easily solved. Thus we have finally,

$$P = (((y+a_1)(y^2+c_2)+\beta_3)(y^2-\alpha_5)) + \beta_5 ,$$
$$\beta_3 = c_3 - a_1c_2.$$

This accomplishes the evaluation with 3 multiplications and 5 additions, provided the quadratic equation could be solved.

Remarks

It can be shown that by solving a cubic equation, *any* fifth degree polynomial can be evaluated with 3 multiplications and 5 additions, and that this is the best possible; the algorithm is:

$z := y + t;$
$w := z \uparrow 2;$
$P := ((w+a)\times w+b) \times (z+c) + d$

where a, b, c, d and t are rather difficult to find (but not any worse than the determination of the constants in the sixth degree algorithm given earlier).

Ostrowski [2] has shown that Horner's Rule minimizes the total number of operations, counting additions on an

equal par with multiplications, for $n \leqq 4$; and we have shown that for $n > 4$ Horner's Rule is never minimal.

Another often important consideration has not been mentioned yet; that is the question of error analysis. Whichever way is chosen to evaluate these polynomials, it should be investigated to see if the proper amount of accuracy is obtained, for the range of y.

The Special Case y^n

Certain polynomials may have special properties which allow them to be evaluated in fewer steps. The eighth degree polynomial of an earlier example required 4 multiplications and 7 additions. And of course, a polynomial where all the coefficients of odd degree vanish requires in general fewer operations.

We consider here only the most special case of all, $P = y^n$. In this case a good compiler should definitely convert this into a minimum number of multiplications somehow. It is a very difficult combinatorial problem, however, to decide how to best compute y^n.

One method which is fairly well known (see, for example, Floyd [3]) and which one might call the Binary Method can be described as follows:

(a) Write n as a number in the binary system; e.g., if $n = (13)_{10}$, $n = (1101)_2$.

(b) Replace each "1" by SX and each "0" by S; e.g., $1101 \rightarrow SXSXSSX$.

(c) Cancel the SX at the left end. The resulting string can be interpreted as instructions: S = "square", X = "multiply by y." In our example we calculate y^{13} with the sequence $SXSSX$.

1. Take y
2. Square (y^2)
3. Multiply by y (y^3)
4. Square (y^6)
5. Square (y^{12})
6. Multiply by y (y^{13}).

We have used 5 multiplications, which in this case can be shown to be minimum.

Another algorithm, believed to be new, might be called the Factor Method. Here we generate sequences $S(n, m)$ as follows:

(a) If n is not prime, $S(n, m) = S(n/p, m)S(p, nm/p)$ where p is the smallest prime factor.

(b) If n is prime, $S(n, m) = S(n-1, m)X_m$.

(c) $S(1, m)$ = null string.

Interpretation: X_m means "multiply by y^m, which is a previously calculated result." $S(n, m)$ is the sequence for the calculation of y^{nm} assuming that y^m has been calculated. The goal is to find $S(n, 1)$.

For example,

$$\begin{aligned} S(13, 1) &= S(12, 1)X_1 = S(6, 1)S(2, 6)X_1 \\ &= S(3, 1)S(2, 3)X_6X_1 \\ &= S(2, 1)X_1X_3X_6X_1 \\ &= X_1X_1X_3X_6X_1 . \end{aligned}$$

This turns out to be identical to the Binary Method in this case.

Another way to describe this algorithm is:

(a) Factor n into primes, $n = p_1p_2 \cdots p_r$.

(b) If n is prime, calculate $(y^{n-1})y$.

(c) If n is composite, calculate $(((y^{p_1})^{p_2}) \cdots)^{p_r}$.

To compare the two methods, the Binary Method uses $r+s-1$ multiplications, where $r = [\log_2 n]$ and s is the number of 1's in the binary representation. The Factor Method uses M_n multiplications, where

$$M_n = M_{n-1} + 1 \quad \text{if } n \text{ is prime,}$$

$$M_n = M_r + M_s \quad \text{if } n = rs \text{ is composite.}$$

The Factor Method is slightly better than the Binary Method overall. For $n \leqq 150$, there are

94 cases where the two methods are equal,
31 cases where the Factor Method is one better,
15 cases where the Binary Method is one better,
9 cases where the Factor Method is two better,
1 case ($n = 129$) where the Binary Method is two better.

The smallest cases where the Factor Method excels are $n = 15, 27, 30, 31^*, 39, 45$. The smallest cases where the Binary Method excels are $n = 33, 49$.

But there are cases where we can do better than *both* methods. For $n \leqq 70$ there are five known cases where this is true. (They are $n = 23, 43, 46, 47, 59$.) A third method, which is the best method known to the author, might be called the Tree Method since it develops a tree.

The Tree Method is a good exercise in list processing techniques. Start at level 0 with a single node labelled 1. To get to the next level when a level is completed, process the nodes on the preceding level from left to right. For each node, try adding successively from the top each of the values above this node. If any new values are obtained, they become nodes branching out to the next level. Duplicate values obtained are discarded.

For example, we might develop three levels as in Figure 1. Discarding values that are duplicates, we get Figure 2. Making the trial additions from top to bottom

FIG. 1

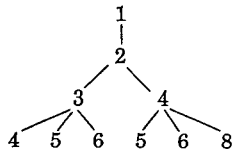


FIG. 2

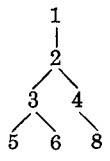
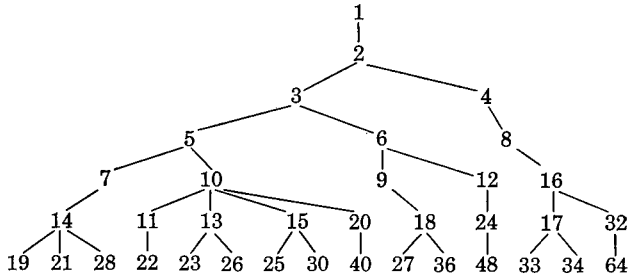


FIG. 3



* See Editor's Note at end of paper.

leads to a much better tree than making them from bottom to top. However, the bottom-to-top method is much more convenient to program, and it can be used if the new branches from a given node are attached from right to left. The tree begins looking as in Figure 3.

The preceding criterion of excellence was only the number of total multiplications. Clearly if n is an *unknown* variable, the Binary Method is the best to use. In fact this method would be quite suitable to incorporate in the hardware of a binary computer, as an exponentiation operator.

If y is floating point, there is of course a point of diminishing returns when n gets large, since it will eventually be better to take logarithms and exponentials. Since the Tree Method uses r multiplications at level r it is possible to stop generating the tree at a certain point and we then have a set of all the "interesting" values of n. The tree in Figure 3, for example, shows all n for which it is known that y^n needs 6 or less multiplications. The Factor Method, on the other hand, is valuable when n is large and an application requires frequent calculation of y^n.

A system manual (which will not be mentioned here by name) has a subroutine for "float to fix exponentiation" of $y \uparrow n$ which uses $n-1$ multiplications since it says "there is small probability of this routine being used with a very large n." This may seem to be a valid point at first; but it didn't take long before a user had to calculate y^{70} a very large number of times, and so the user rewrote the subroutine. This remark is included here to offer some justification for having a good power method.

REFERENCES

1. See TODD, JOHN. *A Survey of Numerical Analysis*, pp. 3-4. McGraw-Hill, 1962.
2. OSTROWSKI, A. M. *Studies in Mathematics and Mechanics Presented to R. von Mises*, pp. 40–48. Academic Press, Inc., 1954.
3. FLOYD, R. An algorithm for coding efficient arithmetic operations. *Comm. ACM 4* (Jan. 1961), 50–51.

EDITOR'S NOTE. Since much usage may be made of these methods in the floating-point mode, one could consider also the relative timing of multiplication and division. In some machines these have even been equal. Thus, for y^{31} other possibilities exist, using "D" to indicate a division:

Binary	(8)	$SXSXSXSX$	$\equiv X_1X_1X_3X_1X_7X_1X_{15}X_1$
Factor	(7)		$X_1X_2X_2X_6X_{12}X_6X_1$
Division	(6)		$X_1X_2X_4X_8X_{16}D_1$

The reader is invited to try n^{127} and see if some algorithm could be derived for mixed operations.

A Decision Matrix as the Basis for a Simple Data Input Routine

G. J. VASILAKOS
Datatrol Corporation, Silver Spring, Md.

Currently a great deal of time and effort is being spent on the development of bigger and better compiler languages, multiprogram executive systems, etc. Since the implementation of of new methods and procedures is not instantaneous, but rather occurs by an evolutionary process, we should be concerned also with the problem of maintaining, improving and incorporating new ideas into existing systems. It is with this somewhat neglected area that the author is interested. A method employing a decision matrix is presented for the handling of a standard systems programming problem, that of providing a data input routine.

Introduction

Motivation for this project came from an analysis of several current systems which revealed that the routines for handling input character data strings had been coded in an ad hoc manner, brute-force, do-it-any-way-you-can method. The technique to be outlined may either suggest that recoding of these programs could be worthwhile or it may at least provide some useful ideas for people designing their own input routines. However, it is not our purpose to suggest a format for a general data input routine. Thus the details of the program to be described, which in itself is fairly simple-minded and somewhat restrictive, are not to be construed as recommended specifications but are provided only for the purposes of illustration.

We are concerned with the analysis of character strings which conform to a fairly universally accepted format for defining input data. Basically this involves categorizing data items as falling into one of three general classes: alphanumeric, hollerith or numeric. Alphanumeric and hollerith information consists of character strings which are translated into the corresponding internal machine representation by the input routine and placed in the computer memory. The distinction is usually made to identify character strings which can be manipulated by the program (alphanumeric), and strings which are fixed (hollerith) and can only be altered by re-assembling or recompiling. Numeric data is converted to an appropriate binary representation (we will restrict ourselves to binary machines) and then placed in memory. An input string can contain any of the three data types mixed in any order. A format statement can be used to communicate information about the organization of the data string to the input routine, or the routine can determine this from the context of the input string.