

Lecture 9: September 30, 2009

CS 330 Discrete Structures
Fall Semester, 2009

1 Polynomial evaluation: how complexity depends on the method

As part of the development of a polynomial approximation to the arctangent function (such as would be required in most compilers), a first-year graduate student in Computer Science at Cornell University¹ was asked to write a program to compute the coefficients of a polynomial given its roots. In other words, values r_1, r_2, \dots, r_n were given, and it was necessary to compute the coefficients a_0, a_1, \dots, a_{n-1} in the n th degree polynomial

$$a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + x^n = (x - r_1)(x - r_2) \cdots (x - r_n).$$

The student remembered, from a high-school course in algebra, that

$$a_k = (-1)^{n-k} \cdot \left[\begin{array}{l} \text{the sum of all } \binom{n}{n-k} \text{ possible prod-} \\ \text{ucts of } n-k \text{ of the roots} \end{array} \right]. \quad (1)$$

We can prove (1) with the kind of argument we used to prove the binomial theorem: We ask how a term x^k can be formed in the unsimplified product $(x - r_1)(x - r_2) \cdots (x - r_n)$. Such a term can occur if k of the n terms contribute their x -term, and the remaining $n - k$ contribute their r -term; there are, of course $\binom{n}{n-k}$ ways to choose the terms that contribute their r -term.

Having come up with a correct algorithm for the problem he faced, the student rushed ahead to write the program implementing the algorithm suggested by equation (1). His behavior was typical—the algorithm is correct and relatively easy to implement, so no thought was given to the *quality* of the algorithm. The program worked beautifully on the various polynomials used as test cases, but when applied to the twelfth and higher degree polynomials for which it was expressly written, it used unreasonable amounts of time. After frittering away much of his time allocation² in this manner, he turned, in desperation, to a professor who specialized in combinatorial analysis and numerical analysis³ for help. The help he received was the following analysis of what he was doing.

How much work was involved in computing a_0 by Equation (1)? In this case (1) simplifies to

$$a_0 = (-1)^n r_1 r_2 \cdots r_n$$

(why?), so that $n - 1$ multiplications of the roots are required. For a_1 ,

$$a_1 = (-1)^{n-1} [r_2 r_3 \cdots r_n + r_1 r_3 r_4 \cdots r_n + \cdots + r_1 r_2 \cdots r_{n-1}];$$

this has $\binom{n}{1}$ terms, each of which requires $n - 2$ multiplications of roots (we ignore multiplying by $(-1)^{n-1}$ because this simply switches the sign). Adding these $\binom{n}{1}$ terms requires $\binom{n}{1} - 1$ additions. Similarly, computing a_{n-1} involves adding $\binom{n}{n-1}$ terms, each of which is simply one of the r_j .

¹Professor Reingold, as it happens.

²Five minutes per semester. Punch cards were issued to the machine operator once in the morning and once in the afternoon, and he fed them to the computer, located off-campus near the airport. He would return printouts of the output at his next trip to campus.

³Professor Robert J. Walker.

In general, computing a_i requires adding $\binom{n}{i}$ terms, each of which is a product of $n - i - 1$ roots; in other words, $\binom{n}{i} - 1$ additions and $\binom{n}{i}(n - i - 1)$ multiplications are needed to get the value of a_i . In total then, the computation of all of the n coefficients requires

$$\begin{aligned} \text{Additions:} & \quad \sum_{i=0}^{n-1} \left[\binom{n}{i} - 1 \right] \\ \text{Multiplications:} & \quad \sum_{i=0}^{n-1} \binom{n}{i} (n - i - 1) \end{aligned}$$

In order to analyze these better, we must first figure out how to evaluate:

$$\sum_{k=0}^n k \binom{n}{k}$$

One way to do this is with the binomial theorem:

$$(1 + x)^n = \sum_{k=0}^n \binom{n}{k} x^k$$

Taking derivatives of both sides with respect to x gives us:

$$n(1 + x)^{n-1} = \sum_{k=0}^n k \binom{n}{k} x^{k-1}$$

Setting $x = 1$ we see that:

$$n2^{n-1} = \sum_{k=0}^n k \binom{n}{k}$$

We may also compute this sum combinatorially. Consider the question of choosing a committee of k people from among n of them, and then electing a chairperson for this committee. There are $\binom{n}{k}$ different ways of choosing the committee, and k ways of electing its chair giving $\sum_{k=0}^n k \binom{n}{k}$ total ways to choose any committee with a chair. This is the left-hand side of the desired identity. Another way to count such committees is to first choose the chairperson, which may be done in n ways, and, for each remaining person, decide individually whether or not that person will be in the committee, which may be done in 2^{n-1} ways by the rule of products, giving a total of $n2^{n-1}$ possibilities. This proves the identity.

We now have the tools to properly analyze the number of additions and multiplications needed for student's implementation of the program. The number of additions is:

$$\begin{aligned} \sum_{i=0}^{n-1} \left[\binom{n}{i} - 1 \right] &= \sum_{i=0}^{n-1} \left[\binom{n}{i} \right] - n \\ &= \sum_{i=0}^n \left[\binom{n}{i} \right] - (n + 1) \\ &= 2^n - (n + 1) \end{aligned}$$

The number of multiplications is:

$$\sum_{i=0}^{n-1} \binom{n}{i} (n - i - 1) = \sum_{i=0}^{n-1} (n - 1) \binom{n}{i} - \sum_{i=0}^{n-1} i \binom{n}{i}$$

$$\begin{aligned}
&= (n-1) \sum_{i=0}^{n-1} \binom{n}{i} - \left(\sum_{i=0}^n i \binom{n}{i} - n \binom{n}{n} \right) \\
&= (n-1)(2^n - 1) - n(2^{n-1} - 1) \\
&= (n-2)2^{n-1} + 1
\end{aligned}$$

Using these formulas we can compute a table of how many arithmetic operations are used:

n	1	2	3	4	5
additions	0	1	4	11	26
multiplications	0	1	5	17	49
n	6	7	8	9	10
additions	57	120	247	502	1,013
multiplications	129	321	769	1,793	4,097
n	11	12	13	14	15
additions	2,036	4,083	8,178	16,369	32,752
multiplications	9,217	20,481	45,057	98,305	212,993

Almost a quarter of million arithmetic operations to get the coefficients of a fifteenth degree polynomial and this does not count auxiliary operations done in loop control, assignment statements, or initialization! At one microsecond per instruction, that is almost a quarter of second; when this process is used inside a loop, enormous amounts of computer time will be required. The truth is that these days, computers can easily handle this amount of computation, but if we then try to solve the problem with $n = 30$, we will square the number of operations to 45,366,018,049...which no computer can handle efficiently.

How *should* the coefficients be computed? The answer is that simple multiplication of polynomials works efficiently. If we have already computed that

$$a_0 + a_1x + a_2x^2 + \cdots + a_{k-2}x^{k-2} + x^{k-1} = (x - r_1)(x - r_2) \cdots (x - r_{k-1}),$$

then

$$\begin{aligned}
(x - r_1)(x - r_2) \cdots (x - r_k) &= (a_0 + a_1x + \cdots + a_{k-2}x^{k-2} + x^{k-1})(x - r_k) \\
&= -r_k a_0 + (a_0 - r_k a_1)x + (a_1 - r_k a_2)x^2 + \cdots + (a_{k-2} - r_k)x^{k-1} + x^k.
\end{aligned}$$

Thus we can use the simple process to compute the coefficients.

```

a[0] := 1
FOR i := 1 TO n DO
  a[i] := 0
FOR k := 1 TO n DO
  COMMENT at this point we have computed
    (x - r_1)(x - r_2) \cdots (x - r_{k-1}) =
    a_0 + a_1x + \cdots + a_{k-2}x^{k-2} + x^{k-1};
    we now multiply this by (x - r_k).
  a[k] := 1
  FOR i := k - 1 TO 1 STEP -1 DO
    a[i] := a[i - 1] - r_k \times a[i]
  a[0] := -r_k \times a[0]

```

Have we made any improvement over the algorithm based on direct calculation from equation (1)? The inner loop executes the statement “ $a[i] := a[i - 1] - r_k \times a[i]$ ” a total of

$$\sum_{k=1}^n \sum_{i=1}^{k-1} 1 = \sum_{k=1}^n (k-1) = \frac{n(n-1)}{2}$$

times. The statement “ $a[0] := -r_k \times a[0]$ ” is executed n times. Thus the algorithm requires $n(n-1)/2$ subtractions and $n(n+1)/2$ multiplications. To compare this with the previous algorithm we note that a subtraction requires the same amount of time as an addition and that in neither case have we counted auxiliary operations such as loop control, assignment statements, initialization, and so on (these will be similar in both cases). Computing a table similar to the one before we find

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
subtractions	0	1	3	6	10	15	21	28	36	45	55	66	78	91	105
additions	1	3	6	10	15	21	28	36	45	55	66	78	91	105	120

Our better algorithm is thus more efficient for $n \geq 4$. Specifically, for a polynomial of degree fifteen, it requires only 0.09% of the time required by the other algorithm. More important, however, is the fact that the new algorithm requires time proportional to n^2 as n gets large, while the first algorithm requires time proportional to $n2^n$, an enormous difference!