A Recurrent Problem

Edward A. Bender writes to correct a problem he discovered in one of Bruce Weide's "recurrences" (recurrence relations) as printed in "A survey of analysis techniques for discrete algorithms," COMPUTING SURVEYS, **9,** 4 (December 1977), 291–313. Bruce Weide replies.—Ed.

Bender's Comment

Weide's paper seems to contain an error in the solution of a recursion. On the left side of page 303 appears the equation

$$T(n) = 2\,T(n/2) + n \log n \qquad (1)$$

Weide observes that if $T(n) = cn \log^2 n$ is used to replace $T(n/2)$ in the right side of (1), then

$$T(n) = cn \log^2 n + (1 - 2c)n \log n + cn. \qquad (2)$$

He then says that since

$$(1 - 2c)n \log n + cn = o(n \log^2 n) \qquad (3)$$

we may conclude that $T(n) = O(n \log^2 n)$,* and, since the coefficient of $n \log^2 n$ is c in (2), we have the better result $T(n) = \Theta(n \log^2 n)$. Although the conclusions are correct, the argument is invalid. To see this, note that with $T(n) = cn \log^r n$ and $r > 1$ in the right side of (1) we obtain

$$T(n) = cn \log^r n + O(n \log^s n) = cn \log^r n + o(n \log^r n), \qquad (4)$$

where $s = \max(1, r - 1) < r$. If Weide's argument were correct, we would have $T(n) = \Theta(n \log^r n)$ for all $r > 1$, an obvious impossibility.

How can this argument be made correct? There is little literature on a general approach to this type of problem. However, the following theorem describes a method commonly used:

THEOREM. Suppose we are given that T satisfies the recursion

$$T(n) = R_n(T(n - 1), \cdots, T(1))$$

where, for sufficiently large n, all Ts that actually appear are positive and R_n is a nondecreasing function of its arguments. If $f(n)$ is positive and satisfies

$$cf(n) \geq R_n(cf(n - 1), \cdots, cf(1)) \qquad (5)$$

for all sufficiently large c, and n, then $T(n) = O(f(n))$. If $g(n)$ is positive and satisfies

$$cg(n) \leq R_n(cg(n - 1), \cdots, cg(1)) \qquad (6)$$

for all sufficiently small positive c and all sufficiently large n, then $T(n) = \Omega(g(n))$.

The theorem is easily proved by induction. Suppose (5) holds. Let m be such that R_n is a nondecreasing function of its arguments whenever $n \geq m$. Let c be the maximum of $T(k)/f(k)$ for $k < m$. The induction hypothesis states that $T(k)/f(k) \leq c$ for $k < n$. By the definition of c, it holds for $n = m$. Using the induction hypothesis for n, (5), and the monotonicity of R_n, we have

$$T(n) = R_n(T(n - 1), \cdots, T(1)) \leq R_n(cf(n - 1), \cdots, cf(1)) \leq cf(n).$$

The proof when (6) holds is the same except that inequalities are reversed and c is the minimum of $T(k)/f(k)$ for all $k < m$ such that $T(k)$ actually appears in some recursion R_n with $n \geq m$. This added constraint on k insures that $c \neq 0$.

To illustrate the theorem, consider again recursion (1). It clearly satisfies the assumptions regarding R_n and the Ts. We can see from (2) that (5) holds whenever $c \geq 1$ and $n \geq 3$. Thus $T(n) = O(n \log^2 n)$. Also, (6) holds whenever $0 < c \leq 1/2$ and $n \geq 3$. Thus $T(n) = \Omega(n \log^2 n)$. Combining these two results we have $T(n) = \Theta(n \log^2 n)$. An examination of the derivation of the "big-oh" term in (4) reveals that for sufficiently large n it is positive if $r > 2$ and negative if $r < 2$. Thus we obtain $T(n) = O(n \log^r n)$ for $r > 2$ and $T(n) = \Omega(n \log^r n)$ for $r < 2$,

---

* Recall that
$f(n) = O(g(n))$ means $f(n)/g(n)$ is bounded as $n \to \infty$,
$f(n) = \Omega(g(n))$ means $g(n)/f(n)$ is bounded as $n \to \infty$,
$f(n) = o(g(n))$ means $f(n)/g(n) \to 0$, and
$f(n) = \Theta(n)$ means $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

and thus eliminate the impossibility obtained earlier.

EDWARD A. BENDER
Mathematics Dept
University of California, San Diego
La Jolla, Calif. 92093

Weide's Response

Bender's treatment of recurrences is more rigorous than it was my objective to use in the paper. The problem is that I did not mention how to deal with the possibility that $c = 0$. The technique I suggested is heuristic; it is intended to help one identify the proposed solution, which can then be proved correct by an inductive argument using the recursion for the inductive step.

The method I proposed is simple. We make an initial guess of the order of the solution, say $T(n) = O(f_0(n))$, and substitute $c_0 f_0(n)$ for $T(n)$ on the right side of the recurrence. This may result in the introduction of lower-order terms $f_1(n), \cdots, f_m(n)$. We then guess that

$$T(n) = c_0 f_0(n) + c_1 f_1(n) + \cdots + c_m f_m(n),$$

which we again substitute on the right side of the recurrence. We continue in this fashion until no new functions appear. This process may require an infinite sequence of functions $f_i(n)$, but this causes no difficulty if we can find a closed-form expression for the infinite sum. Now if the $f_i(n)$ are linearly independent (as they usually are for the recurrences commonly encountered), we can solve the recurrence by equating coefficients of like terms.

If we can find coefficients c_i, with $c_0 \neq 0$, we may conclude correctly that $T(n) = \Omega(f_0(n))$ and, furthermore, that

$$T(n) = \sum_i c_i f_i(n).$$

For the simple example (Bender's equation 1) with the initial guess $T(n) = c_0 n \log^r n$, we must have $c_i = 0$ for all i whenever $r < 2$ or r is nonintegral and larger than 2. This shows that that guess cannot be right. On the other hand, the guess $T(n) = c_0 n \log^k n$ for any integer $k \geq 2$ produces $c_i = 0$ for terms $n \log^k n$ in which the power of log n exceeds 2. This demonstrates that the solution is $T(n) = \Omega(n \log^2 n)$ as proposed. This solution is proved formally by an inductive proof based on the recurrence.

I have not proved that this method always leads to a solution. However, I have studied many cases and the method has never failed. When it does work, it leads to a natural inductive proof that the proposed solution is correct.

BRUCE W WIEDE
Dept. Computer
and Information Science
Ohio State University
Columbus, Ohio 43210

The Real Costs of Software

Irving Wendel writes to inquire about differences between Zelkowitz's description of software development costs [1] and conclusions based on a different study by Alberts [2]. Marvin Zelkowitz replies.—Ed.

Wendel's Comment

Zelkowitz presents an equation that tracks the cost of large scale software projects over time. The equation is translated into a software life cycle graph on page 206. The graph resembles a bell-shaped curve somewhat skewed away from the development phase and toward the maintenance phase. The slope of the curve changes slowly along the entire graph.

Zelkowitz's graph differs substantially from a composite software life cycle graph presented by Alberts [1]. Alberts's graph is based on a 16-year program life cycle. The curve resides slightly above zero for six years of software conceptualization and specification, skyrockets for almost two years of expensive development, and drops sharply to less than 50 percent of its zenith as it enters the maintenance phase, where it declines slowly but steadily for the remaining part of the program's life cycle.

Alberts characterizes the smoothly changing life-cycle graphs as "idealized," theoretical in nature.

Both Alberts and Zelkowitz state that their respective graphs are based on experience. I should be interested to see an explanation of their wide differences.

IRVING K. WENDEL
SWAK—Computer Software Engineering
P O Box 1440
Oakland, Calif 94604