Solving Divide and Conquer Recurrences By Master Theorem á la Charlie

Charlie Obimbo
Dept. of Computing and Information Science
University of Guelph

ABSTRACT

This paper discusses analysis of recursive problems. It delves first on their classification, and then the various methods of solving them, depending on which class the recursive relation belongs to. An improvement on the Master Method is then described and used to demonstrate how this method is used to solve recursive relations on Divide & Conquer problems.

The revised method is found easier to understand and remember when solving recurrences.

Keywords: Algorithms, Complexity, Recurrences, Divide and Conquer, Master Theorem

1. Introduction

Problems that require the solving of recurrence relations occur frequently in Algorithm: Analysis and Design Courses. Some of the algorithms frequently taught in these courses are:

1. Merge Sort	$T(n) = 2T(n/2) + 2(n)$	
2. Towers of Hanoi	$T(n) = 2T(n-1) + 2(1)$	[2]
3. Euclid's Algorithm	$T(n) \leq T(n/2) + 2(1)$	[2]
4. Binary Search	$T(n) = T(n/2) + 2(1)$	[2]
5. Quick Sort	$T(n) \leq dn + \frac{2}{n}\sum_{k=0}^{n-1} T(k)$	
6. Strassen's Method	$T(n) = 7T(n/2) + 2(n^2)$	[3]
7. Large Integer Multiplication	$T(n) = 4T(n/2) + cn;\ T(1) = 0$	[3]

to name but a few.

The indispensable last step when analyzing many algorithms is often to solve a recurrence equation. To effectively teach algorithm analysis, it is therefore essential to teach the students:

1. how to effectively classify the specific type of recurrence. This is dealt with in section 2.2 of this paper.
2. how to use the different methods to solve these problems (in particular, which methods are appropriate for which problems).

2. Recurrence Relations

2.1 Solving Recurrence Relations

There are a number of methods that we frequently use to solve recurrence relations. These are well defined and discussed in popular algorithm analysis texts [1, 2, 3, 4] and even in the Discrete Mathematics text books [5, 6].

Some of the methods discussed in these texts are:

- **The iterative method** (also known as Recursion Tree [1]) which entails expanding the recursive terms on the RHS of a recursive formula by the recursive formula.

  Example: $T(n) = 3T(n\text{-}1) + 2$

  becomes $T(n) = 3[3T(n\text{-}2) + 2] + 2 = 9T(n\text{-}2) + 8 = \ldots$

- **The Intelligent Guess Method**

- **The Master Method**

- **The Method of Characteristic Equations**

2.2 Classification of Recurrence Relations

In most algorithm analysis and design courses taught at undergraduate level, there are three prevalent types of recurrences. There are other types of interesting occurrences, but they may not be relevant to introduce in a short course, since their application would be rare. The essence is to give the students the tools, that they can master, apply and build on as is necessary. For the latter part, there is literature to help.

The three types of recurrences mainly dealt with are:

(a) Recurrences for Divide and conquer algorithms

(b) Homogeneous-linear recurrences with constant coefficients

(c) Homogeneous non-linear recurrences

We will briefly describe these three classes.

2.2.1 Divide & Conquer Algorithms

In a Divide & Conquer some of the typical algorithms covered of this category are:

- Strassen's Matrix Multiplication Algorithm

$$T(n) = \begin{cases} c & n = a \\ 7T(n/2) + \theta(n^2) & n > a \end{cases}$$

  where *a* is the threshold.

- Binary Search

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + \theta(1) & n > 1 \end{cases}$$

- Merge Sort

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + \theta(n) & n > 1 \end{cases}$$

In general, consider an algorithm that divides a problem into *a* subproblems, each of size 1/*b* of the original problem, and the algorithm used to combine the solutions of the subproblems takes *f*(*n*) time. The running time of the algorithm can be expressed as:

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + f(n) & \text{if } n > 1 \end{cases} \tag{2.1}$$

2.2.2 Homogeneous Linear Recurrences

This category of algorithms take the form:

$$T(n) = \begin{cases} T(n_0) & \text{if } n = n_0 \\ a_1T(n-1) + a_2T(n-2) + \cdots + a_2T(n-2) & \text{if } n > n_0 \end{cases} \tag{2.2}$$

For an example, consider the following algorithm, which determines the *n*th fibonacci number recursively:

```
function  Fibrec($n$)
   if $n > 2$  then return $n$
   else return Fibrec($n$-1) + Fibrec($n$-2)
```

It's recursive relation can be expressed as:

$$T(n) = \begin{cases} a & \text{if } n = 0 \text{ or } n = 1 \\ T(n-1) + T(n-2) & \text{if } n > 1 \end{cases}$$

Another example would be the Towers of Hanoi [2]:

```
procedure  Hanoi($m$, $i$, $j$)
    /* Moves the $m$ smallest rings from rod $i$ to rod $j$  */
   if $m > 0$  then
        Hanoi($m$-1, $i$, 6-$i$-$j$)
        write $i$ "$\rightarrow$" $j$
        Hanoi($m$-1, 6-$i$-$j$, $j$)
```

With the recurrence relation:

$$T(m) = \begin{cases} a & \text{if } m = 0 \\ 2T(m-1)+1 & \text{otherwise} \end{cases}$$

The major thing to note in these types of recurrence relations is that, when rewritten, the recurrences have the form:

$$a_0 t_n + a_1 t_{n-1} + \cdots + a_k t_{n-k} = 0$$

- these are called linear homogeneous equations with constant coefficients.
  - they are called linear because they do not contain terms of the form $t_{n-j}t_{n-i}$ or of the form t^2_{n-j}
  - homogeneous because their sum = 0, and
  - with constant coefficients because the *a_i's are constants*.

The solution to a homogeneous recurrence is obtained by using the characteristic equation. This method is well discussed in [2].

2.2.3 Homogeneous non-linear recurrences

The solution to homogeneous recurrences becomes more difficult when the recurrence is non-linear. An example of this is seen in the analysis of the exhaustive method of solving the parenthesization problem in the Matrix-Chain Multiplication problem.

This problem can be defined briefly as follows:

Given a sequence (chain) of *n* matrices, $\langle A_1, A_2, \ldots, A_n \rangle$ we wish to compute the product:

$$A_1 A_2 \ldots A_n$$

We can evaluate this expression using the standard algorithm for multiplying matrices. However, since we know that matrix multiplication is associative, and that multiplying two matrices *A* and *B* of dimensions $(m \times n)$ and $(n \times l)$ requires $m \times n \times l$ multiplications, then the order in which we multiply the matrices matters. This can be seen if we require to multiply the following four matrices: A_1 (4, 500), A_2 (500, 20), A_3 (20, 3000), and A_4 (3000, 2). The differences in the number of multiplications required, depending on the order of multiplying the matrices (parenthesization) is given below:

$(A_1\ A_2)\ (A_3\ A_4)$

$(A_1\ A_2) = A_{12}$	$4 \times 500 \times 20$	=	40 000
$(A_3\ A_4) = A_{34}$	$20 \times 3000 \times 2$	=	120 000
$A_{12}A_{34}$	$4 \times 20 \times 2$	=	160
			160 160

$(A_1\ (A_2\ A_3)\ )\ A_4$

$(A_2\ A_3) = A_{23}$	$500 \times 20 \times 3000$	=	30 000 000
$(A_1\ A_{23}) = A_{13}$	$4 \times 500 \times 3000$	=	6 000 000
$A_{13}A_4$	$4 \times 3000 \times 2$	=	24 000
			36 024 000

As can be seen, the first parenthesization is preferable, as it take only 160 thousand multiplications whereas the second 36 million.

An exhaustive algorithm that checks all possible parenthesizations possible, and then chooses the minimum would have a recurrence of:

$$P(n) = \begin{cases} 1 & \text{if } k = 1 \\ \sum_{k=1}^{n-k} P(k)P(n-k) & \text{if } k > 1 \end{cases}$$

This forms a homogeneous non-linear recurrence relation, the solution of which is similar to the Catalan numbers.

A simpler analysis gives a loose lower bound of 2^n and can easily be solved by induction.

3. The Master Method

The standard method of solving the Divide & Conquer type of recurrences, (equation 2.1 in section 2.2.1):

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + f(n) & \text{if } n > 1 \end{cases} \qquad (3.1)$$

is by using the Master Method (or Master Theorem). This is normally expressed in the following form:

$$\boldsymbol{T(n)} = \left\{ \begin{array}{ll} \Theta\left(\boldsymbol{n}^{\log_b \boldsymbol{a}}\right) & \boldsymbol{f(n)} = \boldsymbol{O}\left(\boldsymbol{n}^{\log_b \boldsymbol{a} - \varepsilon}\right) \\ \Theta\left(\boldsymbol{n}^{\log_b \boldsymbol{a}} \log \boldsymbol{n}\right) & \boldsymbol{f(n)} = \Theta\left(\boldsymbol{n}^{\log_b \boldsymbol{a}}\right) \\ \Theta(\boldsymbol{f(n)}) & \boldsymbol{f(n)} = \Omega\left(\boldsymbol{n}^{\log_b \boldsymbol{a} + \varepsilon}\right) \text{AND} \\ & \boldsymbol{af(n/b)} < \boldsymbol{cf(n)} \text{ for large } \boldsymbol{n} \end{array} \right\} \begin{array}{l} \varepsilon > 0 \\ \boldsymbol{c} < 1 \end{array}$$

This formula is rather complex and difficult to remember. In the next section we state it in a simplified manner. This has been used in the **CIS 3490 (Algorithm: Analysis and Design)** course taught at the University of Guelph, and there has been a significant difference in the performance of the Students who have used it to solve divide and conquer recurrences, using this method.

4. The Master Method á la Charlie

The standard method of solving the Divide & Conquer type of recurrences, (equation 1 in section 2.2.1):

Let
$$T(n) = \begin{cases} d & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + f(n) & \text{if } n > 1 \end{cases}$$

Note that most functions $f(n)$ in such relationships (in the Divide & Conquer algorithms) are bound by polynomials i.e. $f(n) = \theta(n^k)$. Let us, in fact, rewrite this equation as:

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + \theta\left(n^k\right) & \text{if } n > 1 \end{cases} \tag{4.1}$$

Thus in equation (4.1) above, let $c = \log_b a$; then

$$T(n) = \begin{cases} \theta\left(n^k\right) & \text{if } c < k \\ \theta\left(n^k \log n\right) & \text{if } c = k \\ \theta\left(n^c\right) & \text{if } c > k \end{cases}$$

If $f(n)$ is not bound by a polynomial, then $T(n) = \theta(f(n))$.

Example 4.1

Find the asymptotic behaviour of the Strassen's algorithm, which has the following recurrence relation:

$$T(n) = \begin{cases} d & \text{if } n = threshold \\ 7T\left(\frac{n}{2}\right) + hn^2 & \text{if } n > threshold \end{cases}$$

As we can see, here $a = 7$, $b = 2$, $k = 2$.

Thus $c = \log_2 7$. Now since $\log_2 7 > 2$; i.e. $c > k$, then

$$\mathrm{T(n)} = \theta(n^c) = \theta(n^{\log_2 7}) = \theta(n^{2.81})$$

Example 4.2

Find the asymptotic behaviour of the following recurrence relation:

$$T(n) = \begin{cases} 1 & \text{if} \quad n = 1 \\ 5T(n/3) + n^2 & \text{if} \quad n > 1 \end{cases}$$

As we can see, here $a = 5$, $b = 3$, $k = 2$.

Thus $c = \log_3 5$. Now since $\log_3 5 < 2$; i.e. $c < k$, then $T(n) = \theta(n^2)$.

5. Conclusions

The method was taught to students taking the Algorithms: Analysis and Design Course at University of Guelph. There was a very positive response as well as a considerable demonstration of understanding of the recursion material, as compared to the previous year, when the method used was mainly the Master Method as described in Section 3. The students taught the new method, were taught both ways, and a vast majority preferred the new method. They also did significantly better on the questions on recursion.

Here is a sample of the responses I received:

Hi Charlie,

The course was interesting but difficult. But thanks for all the help. Your office hours were really helpful. I liked the Master Theorem a la Charlie. It's very neat and makes it easier in finding the complexities.

Have a good summer.

-A [anonymity for the sake of the Student.]

6. References

1. T. Cormen, C. Leiserson, R. Rivest, C. Stein. Introduction to Algorithms, 2nd Edition. MIT Press (McGraw Hill), 2001.
2. G. Brassard and P. Bratley. Fundamentals of Algorithmics. Prentice Hall, 1996.
3. R. Neapolitan and K. Naimipour. Foundations of Algorithms: with C++ pseudocode, 2nd Edition. Jones and Bartlett Publishers, 1998.
4. U. Manber. Introduction to Algorithms: A Creative Approach. Addison Wesley, 1989.
5. K. Rosen. Discrete Mathematics And Its Applications, 5th Edition. McGraw Hill, 2003.
6. S. Epp. Discrete Mathematics With Applications, 2nd Edition. Brooks/Cole Publishers Company, 1995.

Author

Charlie F. Obimbo

Dept. of Computing and Information Science

University of Guelph

Guelph, ON N1G 2W1

cobimbo@cis.uoguelph.ca

http://www.cis.uoguelph.ca/~cobimbo