Lecture Notes, Fundamental Algorithms: Recurrence Equations

Obtaining recurrence equations

The typical form of a recursive algorithm looks something as follows.

RecProc(n):
if $n \leq base\_val$ **then** base case code C_b
else
non-recursive code C_0;
RecProc(n_1);
non-recursive code C_1;
RecProc(n_2);
non-recursive code C_2;
...
non-recursive code C_{k-1};
RecProc(n_k);
non-recursive code C_k
end if

where $n_1, n_2, \ldots, n_k < n$ are the sizes of the recursive subproblems. Let C_i have worst case runtime t_i, for $1 \leq i \leq k$, and C_b have worst case runtime t_b. Then we can write down the following recurrence equation for $T(n)$, the worst case runtime for RecProc(n).

$$T(n) = \sum_{i=1}^{k} T(n_i) + \sum_{i=1}^{k} t_i \quad n > base\_val$$
$$T(base\_val) = t_b$$

Observe that there is one term on the right hand side for each recursive call.

How to solve them

We illustrate the recursion tree method by example.
Example 1

$$T(n) = 3T(n/2) + cn^2 \quad n \geq 2$$
$$T(1) = c$$

Suppose that $n = 2^k$ for some integer $k \geq 0$.

size of subproblems	number of subproblems	non-recursive cost
$n = 2^k$	1	cn^2
$n/2 = 2^{k-1}$	3	$c \cdot 3(n/2)^2 = cn^2 \cdot (3/4)$
$n/2^2 = 2^{k-2}$	3^2	$c \cdot 3^2(n/2^2)^2 = cn^2 \cdot (3/4)^2$
	...	
$n/2^{k-1} = 2^1$	3^{k-1}	$c \cdot 3^{k-1}(n/2^2)^{k-1} = cn^2 \cdot (3/4)^{k-1}$
$n/2^k = 2^0 = 1$	3^k	$c \cdot 3^k \cdot (n/2^k)^2 = cn^2 \cdot (3/4)^k$

The total cost is

$$\begin{aligned} cn^2\big(1 + 3/4 + (3/4)^2 + \ldots + (3/4)^k\big) &= cn^2[1 - (3/4)^{k+1}]/(1 - 3/4) \\ &= 4cn^2[1 - (3/4)^{k+1}] \\ &= 4cn^2(1 - \frac{3}{4}n^{\log 3/4}). \end{aligned}$$

Check. Base case: $n = 1$, $k = 0$. Our solution gives $T(1) = 4c(1 - 3/4) = c$, which is correct.
The next larger value of n: $n = 2$, $k = 1$. Our solution gives $T(2) = 16c(1 - 9/16) = 7c$, which is also correct.

Example 2 This recurrence yields the number of moves for the Tower of Hanoi problem.

$$\begin{aligned} T(n) &= 3T(n-1) + 1 \quad n \geq 2 \\ T(1) &= 1 \end{aligned}$$

size of subproblems	number of subproblems	non-recursive cost
n	1	1
$n-1$	3	3
$n-2$	3^2	3^2
	...	
$2 = n - (n-2)$	3^{n-2}	3^{n-2}
$1 = n - (n-1)$	3^{n-1}	3^{n-1}

The total cost is $1 + 3 + \ldots + 3^{n-1} = (3^n - 1)/2$.
Check. Base case: $n = 1$. Our solution gives $T(1) = (3 - 1)/2 = 1$, which is correct.
The next larger value of n: $n = 2$. Our solution gives $T(2) = (9 - 1)/2 = 4$, which is also correct.

Example 3

$$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n \quad n \geq 4$$
$$T(2) = 4$$

Suppose that $n = 2^{2^k}$ for some integer $k \geq 0$.

size of subproblems	number of subproblems	non-recursive cost
$n = 2^{2^k}$	1	$n \cdot 1 = n$
$\sqrt{n} = 2^{2^{k-1}}$	$\sqrt{n} = n/2^{2^{k-1}}$	$2^{2^{k-1}} \cdot n/2^{2^{k-1}} = n$
$n^{1/4} = 2^{2^{k-2}}$	$n/2^{2^{k-1}} \cdot 2^{2^{k-1}}/2^{2^{k-2}} = n/2^{2^{k-2}}$	$2^{2^{k-2}} \cdot n/2^{2^{k-2}} = n$
	$\cdots$	
$4 = 2^{2^1}$	$n/2^{2^1}$	$2^{2^1} \cdot n/2^{2^1} = n$
$1 = 2^{2^0}$	$n/2^{2^0}$	$4 \cdot n/2^{2^0} = 2n$

The total cost is $kn + 2n = (k+2)n = n(2 + \log\log n)$.
Check. Base case: $n = 2$, $k = 0$. Our solution gives $T(1) = 2 \cdot 2 = 4$, which is correct.
The next larger value of n: $n = 4$, $k = 1$. Our solution gives $T(4) = 4(2+1) = 12$, which is also correct.

Handling unrestricted values of n

We now explain several ways of addressing the fact that n is not always a convenient value such as 2^k. One approach is to pad the input so as to achieve the special value. If we have an analysis that works for $n = 2^k$, where k is an integer, then we can use the fact that for all positive n there is an integer k such that $2^{k-1} < n \leq 2^k$. So the padding at most doubles the value of n. This does not work well always: if the convenient value is 2^{2^k} then the padding could square the value of k.

We now illustrate how we can use rounding in upper bounding the solution of recurrence equations.

Example 4

$$T(n) = 2T(n/2) + n \quad n \geq 2$$
$$T(1) = 1$$

To begin with, we interpret $n/2$ as meaning $\lceil n/2 \rceil$. Let $k = \lceil \log n \rceil$.

size of subproblems	number of subproblems	non-recursive cost
$n \leq 2^k$	1	$\leq 2^k \cdot 1 = 2^k$
$\leq \lceil n/2 \rceil \leq 2^{k-1}$	$2 = 2^1$	$\leq 2^{k-1} \cdot 2^1 = 2^k$
	$\cdots$	
$\leq 2 = 2^1$	2^{k-1}	$\leq 2^1 \cdot 2^{k-1} = 2^k$
$1 = 2^0$	$\leq 2^k$	$\leq 2^0 \cdot 2^k = 2^k$

The total cost is at most $(k+1)2^k \leq (1 + \lceil \log n \rceil)(2n-1)$. (We are using the fact that $2^{k-1} < n$, and therefore $2^k \leq 2n - 1$.)

Check. Base case: $n = 1$, $k = 0$. Our solution gives $T(1) \leq (k+1)2^k = 1 \geq 1$, and so it is correct. The next larger value of n: $n = 2$, $k = 1$. Our solution gives $T(2) \leq (k+1)2^k = 2 \cdot 2^1 = 4 \geq 4$, which is also correct.

Note that the assurance given by the check is less strong, as we are now verifying an upper bound rather than an equality. But in this case we are still obtaining tight bounds for these values of n. If we used the bound in terms of n, we would obtain larger values, which would still be correct, but less of an assurance.

In the event that the subproblems in the two recursive calls have combined size n, we can give a tighter analysis (this corresponds to one subproblem having size $\lceil n/2 \rceil$ and the other subproblem having size $\lfloor n/2 \rfloor$; this happens in Merge Sort for example). Then, at each level except the last one, the total size of the subproblems will be n, and at the last level, it will be at most n. This yields:

size of subproblems	total size of subproblems	non-recursive cost
$n \leq 2^k$	n	n
$\leq \lceil n/2 \rceil \leq 2^{k-1}$	n	n
	$\cdots$	
$\leq 2 = 2^1$	n	n
$1 = 2^0$	$\leq n$	$\leq n$

The total cost is at most $(k+1)n \leq (1 + \lceil \log n \rceil)n$.
Check. Base case: $n = 1$, $k = 0$. Our solution gives $T(1) \leq (k+1)n = 1 \geq 1$, and so it is correct. The next larger value of n: $n = 2$, $k = 1$. Our solution gives $T(2) \leq (k+1)n = 2 \cdot 2 = 4 \geq 4$, which is also correct.

Example 5 This is a slightly artificial example to show that we can have small excesses in the size of the recursive subproblems and often this will not affect the asymptotic bound.

$$T(n) = T(\lceil (n+1)/2 \rceil) + 1 \quad n \geq 4$$
$$T(3) = 3$$

We still want to choose a parameter k so it decreases by 1 as we go down one level in the recursion. Accordingly, we define k to be the integer satisfying $2^{k-1} + 2 < n \leq 2^k + 2$. Then $\lceil (n+1)/2 \rceil \leq \lceil (2^k + 3)/2 \rceil \leq \lceil 2^{k-1} \rceil + 2$. This yields the following bounds on the costs.

size of subproblems	number of subproblems	non-recursive cost
$n \leq 2^k + 2$	1	1
$\leq 2^{k-1} + 2$	1	1
	$\cdots$	
$\leq 2^1 + 2 = 4$	1	1
$2^0 + 2 = 3$	1	3

The total cost is at most $k + 3 \leq 3 + \log(n-2)$.
Check. Base case: $n = 3$, $k = 0$. Our solution gives $T(3) \leq 3$, and so it is correct.
The next larger value of n: $n = 4$, $k = 1$. Our solution gives $T(4) \leq 4$, which is also correct.