the master method

1 Solving Recurrences
- the cost of divide-and-conquer algorithms
- the recursion tree: depth and #leaves

2 Statement of the Master Theorem
- asymptotic growth: big O, big Omega, and big Theta
- statement and interpretation
- using the master theorem

MCS 360 Lecture 40
Introduction to Data Structures
Jan Verschelde, 24 November 2010

solving recurrences

Solving recurrences consists of two steps:

1. Apply the recursion-tree method for the solution form.
2. Use mathematical induction to find constants in the form and show that the solution works.

The previous lecture dealt with the recursion-tree method, before that we covered the substitution method for step 2.

Today we consider the general case of estimating the cost of divide-and-conquer algorithms.

the master method

divide-and-conquer algorithms

The recurrence relation

$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1, \text{ some function } f.$$

expresses the cost of a recursive algorithm that

- splits a problem of size n in a pieces;
- every piece has size n/b (or $\lfloor n/b \rfloor$, or $\lceil n/b \rceil$); and
- it takes $f(n)$ to divide the problem and assemble the solutions to the pieces.

Example(1): merge sort has cost $T(n) = 2T(n/2) + cn$, for some constant c.

Example(2): $a = 7$ and $b = 2$ in Strassen's matrix multiplication algorithm, and $f(n) = 18n^2$.

the master method

depth of the recursion

The recursion $T(n) = aT(n/b) + f(n)$ stops

- when T is applied to 1,
- at the depth of the recursion tree.

Denote by d the depth of the recursion: $\dfrac{n}{b^d} = 1$.

$$n = b^d \quad \Rightarrow \quad \log_2(n) = \log_2(b^d) = d \log_2(b)$$

We have:

$$d = \log_b(n) = \frac{\log_2(n)}{\log_2(b)}.$$

number of leaves

The recursion $T(n) = aT(n/b) + f(n)$ defines a tree of depth $d = \log_b(n) = \log_2(n)/\log_2(b)$ with L leaves.

At each level #children = a, so $L = a^d$.

$$\begin{aligned} \log_2(L) &= d \log_2(a) \\ &= \frac{\log_2(n)}{\log_2(b)} \log_2(a) \\ &= \frac{\log_2(a)}{\log_2(b)} \log_2(n) \\ &= \log_b(a) \log_2(n) \end{aligned}$$

So the number of leaves L is $n^{\log_b(a)}$.

the master method

Asymptotic Order

big O	:	f is $O(g)$	:	f grows no faster than g
big theta	:	f is $\Theta(g)$	:	f grows at the same rate as g
big omega	:	f is $\Omega(g)$	:	f grows at least as fast as g

Viewed as sets: $\Theta(g) = O(g) \cap \Omega(g)$.

Limit definitions:

- f is $O(g)$ if $\displaystyle\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} < \infty$, including 0
- f is $\Omega(g)$ if $\displaystyle\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} > 0$, including ∞
- f is $\Theta(g)$ if $\displaystyle\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} = c$, $0 < c < \infty$

$\displaystyle\lim_{n \rightarrow +\infty}$ means for all $n \geq N$, for some constant N

the master method

the master theorem

The recurrence relation

$$T(n) = aT(n/b) + f(n), \quad a \geq 1, b > 1, \text{ some function } f$$

has the following bounds:

1. If $f(n)$ is $O(n^{\log_b(a)-\epsilon})$ for some constant $\epsilon > 0$, then $T(n)$ is $\Theta(n^{\log_b(a)})$.
2. If $f(n)$ is $\Theta(n^{\log_b(a)})$, then $T(n)$ is $\Theta(n^{\log_b(a)} \log_2(n))$.
3. If $f(n)$ is $\Omega(n^{\log_b(a)+\epsilon})$, for some constant $\epsilon > 0$, and

   $$a\, f(n/b) \leq c\, f(n), \text{ for some constant } c < 1 \text{ and } n \geq N,$$

   then $T(n)$ is $\Theta(f(n))$.

interpretation of the theorem

Recall: the recurrence $T(n) = aT(n/b) + f(n)$ defines a tree of depth $\log_b(n)$ with $L = n^{\log_b(a)}$ leaves.

Three cases:

1. f grows no faster than L: f is $O(n^{\log_b(a)})$
2. f grows at the same rate as L: f is $\Theta(n^{\log_b(a)})$
3. f grows at least as fast as L: f is $\Omega(n^{\log_b(a)})$

Relating $f(n)$ to L, we find that $T(n)$ depends on L.

Second case: f is $\Theta(L) \Rightarrow T(n)$ is $\Theta(L\log_2(n))$.

Note: depth $\log_b(n) = \log_2(n)/\log_2(b)$ is $\Theta(\log_2(n))$.

the master method

merge sort

The merge sort algorithm has a recurrence

$$T(n) = 2T(n/2) + \Theta(n),$$

where $a = 2$, $b = 2$, and $f(n) = \Theta(n)$.

The number of leaves: $L = n^{\log_b(a)} = n^{\log_2(2)} = n$, so case 2 of the theorem applies:

2. If $f(n)$ is $\Theta(n^{\log_b(a)})$, then $T(n)$ is $\Theta(n^{\log_b(a)} \log_2(n))$.

Therefore: $T(n)$ is $O(n \log_2(n))$.

Strassen's matrix multiplication

For the matrix multiplication algorithm of Strassen we have $a = 7$, $b = 2$, and $f(n) = 18n^2$.

The number of leaves: $L = n^{\log_b(a)} = n^{\log_2(7)}$, and $\log_2(7) \approx 2.80736$.

Which case applies?

Compare growth of f, which is $\Theta(n^2)$ to $O(n^{2.80736})$.

Take $\epsilon = 0.80736$ and case 1 applies:

1. If $f(n)$ is $O(n^{\log_b(a)-\epsilon})$ for some constant $\epsilon > 0$, then $T(n)$ is $\Theta(n^{\log_b(a)})$.

Therefore: $T(n)$ is $O(n^{2.81})$ which is better than $O(n^3)$.

case 3 applies

Let $T(n) = 3T(n/4) + n\log_2(n)$, so $a = 3$, $b = 4$, and $f(n) = n\log_2(n)$.

$L = n^{\log_b(a)} = n^{\log_4(3)}$ is $O(n^{0.793})$, as $\log_4(3) \approx 0.79248$.

Take $\epsilon = 0.2$, then $f(n)$ is $\Omega(n^{\log_4(3)+\epsilon})$.

Does $a\,f(n/b) \leq c\,f(n)$ hold for some constant $c < 1$ asymptotically? We check:

$$a\,f(n/b) = 3(n/4)\log_2(n/4) \leq 3/4n\log_2(n) = c\,f(n),$$

holds for $c = 3/4$.

Therefore: $T(n)$ is $\Theta(n\log_2(n))$.

the master method does not apply

Let $T(n) = 2T(n/2) + n\log_2(n)$, so $a = 2$, $b = 2$, and $f(n) = n\log_2(n)$.

It appears easy at first as $n^{\log_b(a)} = n^{\log_2(2)} = n$.

As $f(n)$ grows faster than n, we are drawn to case 3.

However, verifying the condition $a\,f(n/b) \leq c\,f(n)$:

$$2(n/2)\log_2(n/2) = n(\log_2(n) - 1) = n\log_2(n) - n$$

and

$$n\log_2(n) - n \leq c\,n\log_2(n) \quad \text{or} \quad \log_2(n) - 1 \leq c\,\log_2(n)$$

does not permit a value of $c < 1$.

Summary + Assignments

We covered §4.5 of *Introduction to Algorithms*, 3rd edition by Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson.

Assignments:

1. Consider $T(n) = 2T(n/4) + n^k$ for $k = 0, 1/2, 1, 2$. Solve the recurrence for each k.
2. Use the master method to solve the recurrence for binary search $T(n) = T(n/2) + O(1)$.
3. Can the master method be applied to $T(n) = 4T(n/2) + n^2 \log_2(n)$? Justify your answer.

Last homework collection on Monday 29 November:
#1 of L-30, #1 of L-31, #3 of L-32, #2 of L-33, #1 of L-34.

Final exam on Tuesday 7 December, 8-10AM in TH 216.