Recurrences

David Kauchak

Recurrence: a function that is defined with respect to itself on smaller inputs.

- Why are we concerned with recurrences?

  The computational costs of divide and conquer algorithms and, in general, recurive algorithms, can often be described easily using recurrences.

- The problem?

  Recurrences are easy to define, but they don't readily express the actual computational cost of the algorithm. We want to remove the self-recurrence and determine a more understandable form of the function.

- The methods

  Each approach will provide you with a different way for analyzing recurrences. Depending on the situation, one or more of the approaches may be applicable.

  - Substitution method: When we have a good guess of the solution, we start with that then prove that it is correct
  - Recursion-tree method: If we don't have a good guess of the solution, looking at the recursion tree can help us. Then, we prove it is correct with the substitution method.
  - Master method: Provides solutions for recurrences of the form: $T(n) = aT(n/b) + f(n)$

- The substitution method: Guess the form of the solution. Assume it's correct and show that the solution is appropriate using a proof by induction.

- $T(n) = \begin{cases} d & \text{if } n = 1 \\ T(n) = T(n/2) + d & \text{otherwise} \end{cases}$

  Halves the input at each iteration and does a constant amount of work, e.g. binary search - Guess: $O(\log_2 n)$

  To show that $T(n) = O(\log_2 n)$, we need to find constants c and n_0 such that $T(n) \leq c \log_2 n$ for all $n \geq n_0$

  We'll find the constants and do the proof by induction at the same time.

  Base case:

  * $n = 1$?

  $$\begin{aligned} T(1) = d &\leq c \log_2 1 \\ &\leq c \cdot 0 \quad \textbf{?} \end{aligned}$$

  * $n = 2$?

  $$\begin{aligned} T(2) = 2d &\leq c \log_2 2 \\ &\leq c \end{aligned}$$

    which is true if $c \geq 2d$.

  Inductive case:

  Assume $T(k) \leq c \log k$ for $k < n$ and show $T(n) \leq c \log n$ for some constant $c > 0$.

  $$\begin{aligned} T(n) &= T(n/2) + d \\ &\leq c \log_2 (n/2) + d \qquad \text{(by induction)} \\ &= c \log_2 n - c \log_2 2 + d \\ &= c \log_2 n - c + d \\ &\leq c \log_2 n \end{aligned}$$

  if $c \geq d$. So, for $c \geq 2d$ and $n_0 = 2$, $T(n) \leq c \log_2 n$ for all $n \geq n_0$ so, $T(n) = O(\log_2 n)$

– $T(n) = \begin{cases} d & \text{if } n = 1 \\ T(n) = T(n-1) + n & \text{otherwise} \end{cases}$

At each iteration, iterates over all n, reducing the size by one element at each step, e.g. INSERTION-SORT - $O(n^2)$

Base case:

$n = 1$?

$$\begin{aligned} T(1) = d &\leq c1^2 \\ &= c \end{aligned}$$

which is true if $c \geq d$

Inductive step:

Assume $T(k) \leq ck^2$ for $k < n$ and show $T(n) \leq cn^2$ for some constant $c > 0$.

$$\begin{aligned} T(n) &= T(n-1) + n \\ &\leq c(n-1)^2 + n \\ &= c(n^2 - 2n + 1) + n \\ &= cn^2 - 2cn + c + n \\ &\leq cn^2 \end{aligned}$$

if

$$\begin{aligned} -2cn + c + n &\leq 0 \\ -2cn + c &\leq -n \\ c(-2n + 1) &\leq -n \\ c &\geq \frac{n}{2n-1} \\ c &\geq \frac{1}{2 - 1/n} \end{aligned}$$

which is true for any $c \geq 1$ for $n \geq 1$. So, for $c \geq d$ (assuming $d \geq 1$) and $n_0 = 1$, then $T(n) \leq cn^2$ for all $n \geq n_0$, so $T(n) = O(n^2)$.

- $T(n) = 2T(n/2) + n$
  Recurses into 2 sub-problems that are half the size and performs some operation on all of the elements, e.g. MERGE-SORT - $O(n \log n)$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\leq 2cn/2 \log(n/2) + n \\ &= 2cn/2 \log n - 2cn/2 \log 2 + n \\ &\leq cn \log n - cn + n \end{aligned}$$

  if $cn \geq n$, i.e. $c \geq 1$
- Some other tricks
  * Lower order constants
  * Changing variables

- Recursion-tree method

  Sometimes it is difficult to guess the correct answer to the recurrence. We can look at the tree of recursion calls to get at the correct answer.

  $T(n) = 3T(n/4) + n^2$

  Recursion tree:

  - level 0 - cn^2
  - level 1 - $c(\frac{n}{4})^2 + c(\frac{n}{4})^2 + c(\frac{n}{4})^2 = c\frac{3}{16}n^2$
  - level 2 - $c(\frac{n}{16})^2 ... = c(\frac{3}{16})^2 n^2$
  - level d - $c(\frac{3}{16})^d n^2$

  What is the depth of the tree?

  The end of the recursion occurs when:

$$\begin{aligned} n/4^d &= 1 \\ \log(n/4^d) &= 0 \\ \log n - \log 4^d &= 0 \\ \log n - d \log 4 &= 0 \\ \log_4 n - d &= 0 \\ d &= \log_4 n \end{aligned}$$

What is the cost of the final level?

$T(1)$ for each node and there are

$$
\begin{aligned}
3^d &= 3^{\log_4 n} \\
&= 4^{\log_4 3^{\log_4 n}} \\
&= 4^{\log_4 n \log_4 3} \\
&= 4^{\log_4 n^{\log_4 3}} \\
&= n^{\log_4 3}
\end{aligned}
$$

leaves. For a total cost of $\theta(n^{\log_4 3})$ at the bottom level.

The sum of the costs of the entire tree is the cost of the recurrence relation.

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 ... + (\frac{3}{16})^{d-1} + \theta(n^{\log_4 3}) \\
&= cn^2 \sum_{i=0}^{\log_4 n - 1} (\frac{3}{16})^i + \theta(n^{\log_4 3}) \\
&= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \theta(n^{\log_4 3})
\end{aligned}
$$

where we obtain the last line from $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$ and let $x = \frac{3}{16}$ and $k = \log_4 n - 1$

- Master method - Provides solutions to recurrences of the form $T(n) = aT(n/b) + f(n)$

  Many different versions out there ([3] pg. 49)

  $T(n) = aT(n/b) + O(n^d)$

$$
T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}
$$

  The one we'll use:

$T(n) = aT(n/b) + f(n)$

- if $f(n) = O(n^{\log_b a - \epsilon})$ for $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
- if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
- if $f(n) = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon > 0$ and $af(n/b) \leq cf(n)$ for $c < 1$ then $T(n) = \Theta(f(n))$

- Examples (adapted from [1])
  - $T(n) = 16T(n/4) + n$
    $a = 16$
    $b = 4$
    $f(n) = n$

    $$\begin{aligned} n^{\log_b a} &= n^{\log_4 16} \\ &= n^2 \end{aligned}$$

    Is $f(n) = O(n^{2-\epsilon})$?
    Is $f(n) = \Theta(n^2)$?
    Is $f(n) = \Omega(n^{2+\epsilon})$?

    Case 1: $\Theta(n^2)$
  - $T(n) = T(n/2) + 2^n$
    $a = 1$
    $b = 2$
    $f(n) = 2^n$

    $$\begin{aligned} n^{\log_b a} &= n^{\log_2 1} \\ &= n^0 \end{aligned}$$

    Is $f(n) = O(n^{0-\epsilon})$?
    Is $f(n) = \Theta(n^0)$?
    Is $f(n) = \Omega(n^{0+\epsilon})$?
        Is $2^{n/2} \leq c2^n$?

    Case 3: $\Theta(2^n)$

- $T(n) = 2T(n/2) + n$
  $a = 2$
  $b = 2$
  $f(n) = n$

$$\begin{aligned} n^{\log_b a} &= n^{\log_2 2} \\ &= n \end{aligned}$$

  Is $f(n) = O(n^{1-\epsilon})$?
  Is $f(n) = \Theta(n^1)$?
  Is $f(n) = \Omega(n^{1+\epsilon})$?
  Case 2: $n \log n$

- $T(n) = 16T(n/4) + n!$
  $a = 16$
  $b = 4$
  $f(n) = n!$

$$\begin{aligned} n^{\log_b a} &= n^{\log_4 16} \\ &= n^2 \end{aligned}$$

  Is $f(n) = O(n^{2-\epsilon})$?
  Is $f(n) = \Theta(n^2)$?
  Is $f(n) = \Omega(n^{2+\epsilon})$?
  Is $16(n/4)! \leq cn!$ for all sufficiently large n?

  Case 3: $\Theta(n!)$

- $T(n) = \sqrt{2}T(n/2) + logn$
  $a = 2^{\frac{1}{2}}$
  $b = 2$
  $f(n) = \log n$

$$\begin{aligned} n^{\log_b a} &= n^{\log_2 2^{\frac{1}{2}}} \\ &= n^{\frac{1}{2}} \\ &= \sqrt{n} \end{aligned}$$

Is $f(n) = O(n^{.5-\epsilon})$?
Is $f(n) = \Theta(n^{.5})$?
Is $f(n) = \Omega(n^{.5+\epsilon})$?
Case 1: $\Theta(\sqrt{n})$

- $T(n) = 4T(n/2) + n$
  $a = 4$
  $b = 2$
  $f(n) = n$

$$\begin{aligned} n^{\log_b a} &= n^{\log_2 4} \\ &= n^2 \end{aligned}$$

  Is $f(n) = O(n^2)$?
  Is $f(n) = \Theta(n^2)$?
  Is $f(n) = \Omega(n^{2+\epsilon})$?
  Case 1: $\Theta(n^2)$

These notes are adapted from material found in chapter 4 [2].

References
[1] http://www.csd.uwo.ca/∼moreno//CS424/Ressources/master.pdf
[2] Thomas H. Cormen, Charles E. Leiserson Ronald L. Rivest and Clifford Stein. 2007. Introduction to Algorithms, 2nd ed. MIT Press.
[3] Sanjoy Dasgupta, Christos Papadimitiou and Umesh Vazirani. 2008. Algorithms. McGraw-Hill Companies, Inc.