Objectives for Tue 10/3/2023

- Recurrence relations
- Master theorem

Recurrence relations

Recurrence relation is a mathematical function that is defined in terms of itself, usually with some initial conditions (like base cases)

Example: Fibonacci

Fib(n) = Fib(n - 1) + Fib(n - 2)
Fib(0) = 0
Fib(1) = 1

Divide and conquer problems

- Binary search
- Merge sort
- Quicksort
- Selection (using quicksort partition)
- Closest pair of points in a set (on x-y plane)
- Matrix multiplication → Strassen's algorithm
- Fast Fourier Transform (FFT) - Cooley Tukey

O(…) is not a function, but…

If we say something like…

$$f(n) = 3n + O(1)$$

…what we really mean is…

$$f(n) = 3n + g(n)$$

… where g(n) *is* O(1).

In this case, g(n) need not even be a constant.

Example: $g(n) = \min(|n|, 5)$ // $g \in [0, 5]$

Binary search

Let T(n) = the time to search for an arbitrary element in an array of size n.

T(n) is the average-case time to search for an arbitrary element in an array of size *n*.

$$T(n) = T(n/2) + O(1)$$

$$T(0) = 0$$

$$T(1) = 0$$

In the above equation, O(1) is shorthand for some function g(n) where g(n) is O(1).

Master theorem (for algorithms)

SIGACT News 36 Fall 1980

A General Method for Solving Divide-and-Conquer Recurrences[1]

Jon Louis Bentley[2]
Dorothea Haken
James B. Saxe
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

The approximate complexity of divide-and-conquer algorithms is often described by recurrence relations of the form

$$T(n) = kT(n/c) + f(n) .$$

The only well-defined method currently used for solving such recurrences consists of solution tables for fixed functions f and varying k and c. In this note we describe a unifying method for solving these recurrences that is both general in applicability and easy to apply. This method is appropriate both as a classroom technique and as a tool for practicing algorithm designers.

1. Introduction

The mathematical analysis of algorithms has proven to be an important subject of both practical and theoretical interest. One school within this field, championed by Professor D. E. Knuth of Stanford University, has concentrated on *exact* analyses. In addition to providing practical tools (i.e., the algorithms analyzed), this research has also produced much deep and beautiful mathematics. On the other hand, *approximate* analyses, although less substantial mathematically, are still of practical

Divide-and-Conquer algorithms

Given an algorithm of this form:

```
procedure p(input x of size n):
    if n < some constant k:
        Solve x directly without recursion
    else:
        Create a subproblems of x, each having size n/b
        Call procedure p recursively on each subproblem
        Combine the results from the subproblems
```

Runtime is given by this recurrence relation:

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

a (constant) is the number of subproblems in the recursion

b (constant) is the factor by which the n is reduced in each recursive call

f(n) is the time to create the subproblems and combine their results

Credit: Wikipedia "Master theorem (analysis of algorithms)"

Master theorem (for algorithms)

$$T(n) = aT\left\lceil\frac{n}{b}\right\rceil + f(n)$$

$f(n)$ is a function that is $O(n^d)$.

$a > 0$ // because we recurse at least once

$b > 1$ // because we reduce input at each step

$d \geq 0$ // because <1 unit of work would make no sense

a, b, and d are constants.

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

$c_{\text{crit}} = \log_b a = \log(\#\text{subproblems}) / \log(\text{relative subproblem size})$ Credit: Wikipedia "Master theorem (analysis of algorithms)"

f(n) and d

- f(n) is the time to do the work in the procedure, excluding any recursive calls.
  - splitting into subproblems
  - combining results
  - comparing values
  - deciding which subproblem(s) to solve in the next recursive call(s)
- f(n) is $O(n^d)$
- d is the exponent (above)
- d=1 indicates the work is O(n)
  - Ex: merge step of merge sort
  - Ex: partition in quicksort or our selection algorithm
- d=0 indicates the work is O(1)
  - Ex: comparing search value with middle value in array in binary search

Tighter version…

- Wikipedia has a tighter version, which gives Big-Theta bounds and uses the k parameter.
- $c_{crit} = \log_b a$ in the Wikipedia version.
- We will stick with the simpler version on the previous slide.

Master theorem (for algorithms)

Case	Description	Condition on $f(n)$ in relation to c_{crit}, i.e. $\log_b a$	Master Theorem bound	Notational examples
1	Work to split/recombine a problem is dwarfed by subproblems. i.e. the recursion tree is leaf-heavy	When $f(n) = O(n^c)$ where $c < c_{\text{crit}}$ (upper-bounded by a lesser exponent polynomial)	... then $T(n) = \Theta\left(n^{c_{\text{crit}}}\right)$ (The splitting term does not appear; the recursive tree structure dominates.)	If $b = a^2$ and $f(n) = O(n^{1/2-\epsilon})$, then $T(n) = \Theta(n^{1/2})$.
2	Work to split/recombine a problem is comparable to subproblems.	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for a $k \geq 0$ (rangebound by the critical-exponent polynomial, times zero or more optional logs)	... then $T(n) = \Theta\left(n^{c_{\text{crit}}} \log^{k+1} n\right)$ (The bound is the splitting term, where the log is augmented by a single power.)	If $b = a^2$ and $f(n) = \Theta(n^{1/2})$, then $T(n) = \Theta(n^{1/2} \log n)$. If $b = a^2$ and $f(n) = \Theta(n^{1/2} \log n)$, then $T(n) = \Theta(n^{1/2} \log^2 n)$.
3	Work to split/recombine a problem dominates subproblems. i.e. the recursion tree is root-heavy.	When $f(n) = \Omega(n^c)$ where $c > c_{\text{crit}}$ (lower-bounded by a greater-exponent polynomial)	... this doesn't necessarily yield anything. Furthermore, if $af\left(\frac{n}{b}\right) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n (often called the *regularity condition*) then the total is dominated by the splitting term $f(n)$: $T(n) = \Theta(f(n))$	If $b = a^2$ and $f(n) = \Omega(n^{1/2+\epsilon})$ and the regularity condition holds, then $T(n) = \Theta(f(n))$.

$$c_{\text{crit}} = \log_b a = \log(\#\text{subproblems}) / \log(\text{relative subproblem size})$$

Credit: Wikipedia "Master theorem (analysis of algorithms)"

Master theorem (for algorithms)

A useful extension of Case 2 handles all values of k:[3]

Case	Condition on $f(n)$ in relation to c_{crit}, i.e. $\log_b a$	Master Theorem bound	Notational examples
2a	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for any $k > -1$	... then $T(n) = \Theta\left(n^{c_{\text{crit}}} \log^{k+1} n\right)$ (The bound is the splitting term, where the log is augmented by a single power.)	If $b = a^2$ and $f(n) = \Theta(n^{1/2} / \log^{1/2} n)$, then $T(n) = \Theta(n^{1/2} \log^{1/2} n)$.
2b	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for $k = -1$	... then $T(n) = \Theta\left(n^{c_{\text{crit}}} \log \log n\right)$ (The bound is the splitting term, where the log reciprocal is replaced by an iterated log.)	If $b = a^2$ and $f(n) = \Theta(n^{1/2} / \log n)$, then $T(n) = \Theta(n^{1/2} \log \log n)$.
2c	When $f(n) = \Theta(n^{c_{\text{crit}}} \log^k n)$ for any $k < -1$	... then $T(n) = \Theta\left(n^{c_{\text{crit}}}\right)$ (The bound is the splitting term, where the log disappears.)	If $b = a^2$ and $f(n) = \Theta(n^{1/2} / \log^2 n)$, then $T(n) = \Theta(n^{1/2})$.

Credit: Wikipedia "Master theorem (analysis of algorithms)"

Master theorem (for algorithms)

Application to common algorithms [edit]

Algorithm	Recurrence relationship	Run time	Comment
Binary search	$T(n) = T\left(\frac{n}{2}\right) + O(1)$	$O(\log n)$	Apply Master theorem case $c = \log_b a$, where $a = 1, b = 2, c = 0, k = 0$ [5]
Binary tree traversal	$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$	$O(n)$	Apply Master theorem case $c < \log_b a$ where $a = 2, b = 2, c = 0$[5]
Optimal sorted matrix search	$T(n) = 2T\left(\frac{n}{2}\right) + O(\log n)$	$O(n)$	Apply the Akra–Bazzi theorem for $p = 1$ and $g(u) = \log(u)$ to get $\Theta(2n - \log n)$
Merge sort	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log n)$	Apply Master theorem case $c = \log_b a$, where $a = 2, b = 2, c = 1, k = 0$

Credit: Wikipedia "Master theorem (analysis of algorithms)"