Solving Recurrence Relations

Motivation

We frequently have to solve recurrence relations in computer science.

For example, an interesting example of a heap data structure is a Fibonacci heap. This type of heap is organized with some trees. It's main feature are some lazy operations for maintaining the heap property. Analyzing the amortized cost for Fibonacci heaps involves solving the Fibonacci recurrence. We will outline a general approach to solve such recurrences.

The running time of divide-and-conquer algorithms requires solving some recurrence relations as well. We will review the most common method to estimate such running times.

Linear Hom. Recurrence Relations

A linear homogeneous recurrence relation of degree k with constant coefficients is a recurrence relation of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k},$$

where $c_1, \ldots, c_k$ are real numbers, and $c_k \neq 0$.

linear: a_n is a linear combination of a_k's
homogeneous: no terms occur that aren't multiples of a_k's
degree k: depends on previous k coefficients.

Example

$f_n = f_{n-1} + f_{n-2}$ is a linear homogeneous recurrence relation of degree 2.

$g_n = 5\, g_{n-5}$ is a linear homogeneous recurrence relation of degree 5.

$g_n = 5\, g_{n-5} + 2$ is a linear inhomogeneous recurrence relation.

$g_n = 5\, (g_{n-5})^2$ is a nonlinear recurrence relation.

Remark

Solving linear homogeneous recurrence relations can be done by generating functions, as we have seen in the example of Fibonacci numbers.

Now we will distill the essence of this method, and summarize the approach using a few theorems.

Fibonacci Numbers

Let F(x) be the generating function of the Fibonacci numbers.

Expressing the recurrence $f_n = f_{n-1}+f_{n-2}$ in terms of F(x) yields

$F(x) = xF(x)+x^2F(x)$ + corrections for initial conditions

(the correction term for intial conditions is given by x).

We obtained: $F(x)(1-x-x^2) = x$ or $F(x) = x/(1-x-x^2)$

We factored $1-x-x^2$ in the form $(1-a_1x)(1-a_2x)$ and expressed the generating function F(x) as a linear combination of

$1/(1-a_1x)$ and $1/(1-a_2x)$

A Point of Confusion

Perhaps you might have been puzzled by the factorization

$$p(x) = 1-x-x^2 = (1-\alpha_1 x)(1-\alpha_2 x)$$

Writing the polynomial p(x) backwards,

$$c(x) = x^2 p(1/x) = x^2-x-1 = (x-\alpha_1)(x-\alpha_2)$$

yields a more familiar form. We will call c(x) the characteristic polynomial of the recurrence $f_n = f_{n-1}+f_{n-2}$

Characteristic Polynomial

Let

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

be a linear homogeneous recurrence relation. The polynomial

$$x^k - c_1 x^{k-1} - \cdots - c_{k-1} x - c_k$$

is called the characteristic polynomial of the recurrence relation.

Remark: Note the signs!

Theorem

Let c_1, c_2 be real numbers. Suppose that

$$r^2 - c_1 r - c_2 = 0$$

has two distinct roots r_1 and r_2. Then a sequence (a_n) is a solution of the recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2}$$

if and only if

$$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$$

for $n \geq 0$ for some constants α_1, α_2.

Idea of the Proof

The proof proceeds exactly as in the case of the Fibonacci numbers. Try to prove it yourself!

You might have noticed that it was assumed that the two roots of the characteristic polynomial are the not the same.

Example

Solve the recurrence system

$$a_n = a_{n-1} + 2a_{n-2}$$

with initial conditions $a_0 = 2$ and $a_1 = 7$.

The characteristic equation of the recurrence is

$$r^2 - r - 2 = 0.$$

The roots of this equation are $r_1 = 2$ and $r_2 = -1$. Hence, (a_n) is a solution of the recurrence iff

$$a_n = \beta_1 2^n + \beta_2 (-1)^n$$

for some constants β_1 and β_2. From the initial conditions, we get

$$\begin{aligned} a_0 &= 2 = \beta_1 + \beta_2 \\ a_1 &= 7 = \beta_1 2 + \beta_2 (-1) \end{aligned}$$

Solving these equations yields $\beta_1 = 3$ and $\beta_2 = -1$. Hence,

$$a_n = 3 \cdot 2^n - (-1)^n.$$

Further Reading

Our textbook discusses some more variations of the same idea. For example:

- How to solve recurrences which have characteristic equations with repeated roots

- How to solve recurrence of degree > 2

- How to solve recurrences of degree > 2 with repeated roots.

- How to solve certain inhomogeneous recurrences.

Divide-and-Conquer Algorithms and Recurrence Relations

Divide-and-Conquer

Suppose that you wrote a recursive algorithm that divides a problem of size n into

- a subproblems,

- each subproblem is of size n/b.

Additionally, a total of $g(n)$ operations are required to combine the solutions.

How fast is your algorithm?

Divide-and-Conquer Recurrence

Let f(n) denote the number of operations required to solve a problem of size n. Then

$f(n) = a\, f(n/b) + g(n)$

This is the divide-and-conquer recurrence relation.

Example: Binary Search

Suppose that you have a sorted array with n elements. You want to search for an element within the array. How many comparisons are needed?

Compare with median to find out whether you should search the left n/2 or the right n/2 elements of the array. Another comparison is needed to find out whether terms of the list remain.

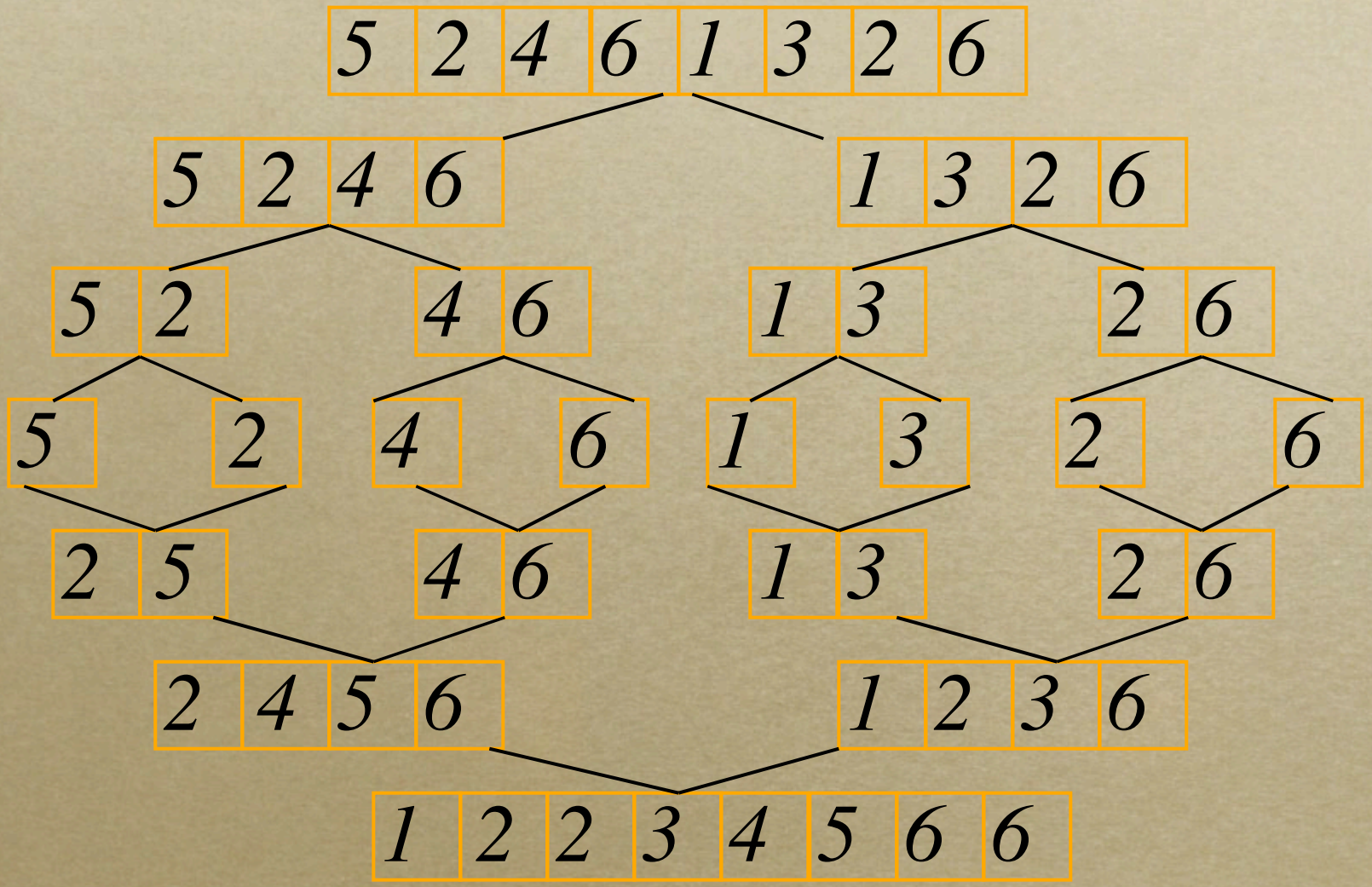
Thus, if f(n) is the number of comparisons, then

$$f(n) = f(n/2) + 2$$

Example: Mergesort

- DIVIDE the input sequence in half
- RECURSIVELY sort the two halves
  - basis of the recursion is sequence with 1 key
- COMBINE the two sorted subsequences by merging them

Mergesort Example



Recurrence Relation for

- Let T(n) be worst case time on a sequence of n keys
- If n = 1, then $T(n) = \Theta(1)$ (constant)
- If n > 1, then $T(n) = 2\ T(n/2) + \Theta(n)$
  - two subproblems of size n/2 each that are solved recursively
  - $\Theta(n)$ time to do the merge

Theorem

Let $f(n)$ be an increasing function satisfying the recurrence

$$f(n) = af(n/b) + c$$

whenever n is divisible by b, $a \geq 1$, $b > 1$ an integer, and c a positive real number. Then

$$f(x) = \begin{cases} O(n^{\log_b a}) & \text{if } a > 1 \\ O(\log n) & \text{if } a = 1 \end{cases}$$

Proof

Suppose that $n = b^k$ for some positive integer k.

$$
\begin{aligned}
f(n) &= af(n/b) + g(n) \\
&= a^2 f(n/b^2) + ag(n/b) + g(n) \\
&= a^3 f(n/b^3) + a^2 g(n/b^2) + ag(n/b) + g(n) \\
&\vdots \\
&= a^k f(n/b^k) + \textstyle\sum_{j=0}^{k-1} a^j g(n/b^j)
\end{aligned}
$$

Suppose that $n = b^k$. For $g(n) = c$, we get

$$f(n) = a^k f(1) + \sum_{j=0}^{k-1} a^j c.$$

For $a = 1$, this yields

$$f(n) = f(1) + ck = f(1) + c\log_b n = O(\log n).$$

For $a > 1$, this yields

$$f(n) = a^k f(1) + c\frac{a^k - 1}{a - 1} = O(n^{\log_b a}).$$

If n is not a power of b, then estimate with the next power of b to get the claimed bounds.